A CLASS OF FORMAL MODELS FOR TRANSLATIONS

A Thesis Submitted

In Partial Fulfilment of the Requirements

For the Degree of

MASTER OF TECHNOLOGY IN ELECTRICAL ENGINEERING

by

VIJAY KUMAR VAISHNAVI

to the

Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

May, 1971

to my parents

CERTIFICATE

This is to certify that the thesis entitled 'A Class of Formal Models for Translations' is a record of the work carried out under my supervision and that it has not been submitted elsewhere for a degree.

H.V. Sahasrabuddhe

(H.V. Sahasrabuddhe)
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

ACKNOWLEDGEMENTS

I am deeply indebted to my thesis advisor, Dr. H.V. Sahasrabuddhe for sustained help and encouragement throughout this work. He has throughout remained a constant source of inspiration to me.

I am grateful to Dr. S.K. Basu for many, useful and stimulating discussions.

I owe my thanks to my friend Mr. B.K. Kaul for sparing his precious time for drawing figures.

Finally, I should thank Mr. J. K. Misra for his excellent typing.

Vijay Kumar Vaishnavi

ABSTRACT

A new class of formal models for translations named n-syntax directed translation grammars is proposed; the models in this class seem to be sufficiently simple as well as powerful. Some of the properties of the range languages of n-syntax directed translation grammars named n-context free languages are investigated. It is shown that n-context free languages define an infinite proper hierarchy of languages with context free languages at one end of the hierarchy.

CONTENTS

INTRODUCTION

There has been considerable interest recently in the formal specification of translations. Some work has been done in the past decade for formalising translation from higher level programming languages to assembly languages of computers. Formal specification of translations between natural languages, however, seems still a difficult task.

A compiler writing system like any translation system in general should be based on a model for the specification of the source and the target languages and the specification of the translation between the two languages. The model should ideally be such that,

a) The translation from any programming language to any assemb language can be completely defined in terms of this model.

b) The overall system based on this model is efficient.

c) The model is simple enough so that it is easy to specify any programming language and any assembly language and the translation between the two.

The models presently available are either too complicated making it almost impossible for an ordinary user to use them besides making the system based on them rather inefficient, or they are not powerful enough to be of much practical use.

This is one of the reasons why compiler writing systems are not in popular use yet. Still the work done so far in this area has already had an impact in compiler writing. An example of this impact is that one common technique in the construction of compilers, presently in use, is to specify the source language by means of a context-free grammar and then define its translation into the object language in terms of this grammar. Compilers utilizing this principle are termed syntax directed, and several current compilers fall into this category (4)[1].

In order to see what is lacking in the presently available models, let us briefly review how the specification of the source and object languages and the translation between the two is done in the presently available models.

It is now generally accepted that a context free grammar (CFG) is a reasonable and concise formalism for the specification of most of the structure of a programming language. Many existing compiler writing systems require that the source language be specified by means of a (usually) restricted) context free grammar(Certain constraints on the source language, such as proper declaration of identifiers or proper use of 'go to' statements cannot be included in the context free specification of the language. Therefore most compiler writing systems allow the use of symbol tables in the specification of the source language. This aspect of compiler writing can be formalised. A formalism has been suggested in (11) whereby context free grammars are augmented by symbol table

1. Numbers in round brackets indicate the reference given at the end of the thesis.

The specification of the object program is usually made in terms of the parse of the source program, and various compiler writing systems have formalisms for specifying the source code into object code.

A simpler formalism for describing translations on a context free grammar is the syntax directed translation grammar (SDTG)(2,8,9,10). Here, associated with each production of the underlying CFG is a rule for permuting the order of the nonterminals on the right side of the production and introducing output symbols on the right side. Given a parse tree in the CFG, with a certain production used at some node, the tree is altered at that node by:

(i) deleting descendants with terminal labels,

(ii) reordering the nonterminal descendants according to the fixed rule, and

(iii) introducing descendants labelled by output symbols.

The translation of a given input sentence is thus produced by parsing the input, performing the above operation at each node of the tree and taking the yield of the resulting tree as output.

A subclass of syntax directed translation grammar, called simple syntax directed translation grammar (SSDTG), allows no permutation of the order of nonterminals on the right side of a production (8).

The above two formalisms (SDTG and SSDTG) though very simple, are not powerful enough to define a large class of translations. Various people have therefore proposed formalisms which are generalisations of the syntax directed translation grammar, so as to make them more powerful devices (3,9,10).

Generalised syntax directed translations (GSDT) is the result of one of these generalisations (3). Here, after producing a parse tree for the input a set of semantic variables (whose values are strings) are evaluated at each node of the tree. The value of a semantic variable at a given node depends only on the production used at that node and the value of semantic variables at its nonterminal descendants. It (the value) is found from these (descendant) variables and constant strings by concatenation. One variable, defined at the root, represents the output.

A formalism which can be considered as a subclass of GSDT is the T_1 semantics (9,10). T_1 semantics are GSDT's with only one semantic variable defined at each node.

These generalisations make the formalism for defining translations quite powerful but the simplicity of the SDT's is lost to a large extent.

Looking back, we see that the formalism for describing the translation from the source to the object language particularly requires serious attention. We would like to have a formalism

whose simplicity is nearer to that of SDT's but which is at the same time more powerful so that atleast some of the features of real life translations (from programming languages to assembly languages) can be described through it.

To make the above idea more specific, let us indicate the type of translations that the model or formalism should be able to describe.

All the programs in the source language comprise of a read statement, an arithmetic expression statement and a punch statement. The BNF description of the source language is as follows:

<program> :: = <read statement> # <a.e.> # <punch statement>#END

<read statement> :: = READ∅<list>

<list> :: = <var>,<list>

<list> :: = <var>

<a.e.> :: = <var>=<expression>

<expression> :: = <var>|<expression><op><var>

<var> :: = A|B|...|E

<op> :: = -|*

<punch statement> :: = PUNCH∅<var>

The object language has besides other op-codes, the op-codes:

ORG, DEFINE, READ, PUNCH, LOAD, ST∅, MPY, SUB, END.

These op-codes have only one operand except the op-code END which has no operand. There is one accumulator.

DORG␣400

means: Load the program in the computer starting at location 400 (the above statement should be at the start of every program).

DEFINE␣A

means: Reserve proper locations for the variable A (The assembler takes care of the rest).

READ␣A

means: Read input starting at location A.

PUNCH␣A

means: Punch the data located at A.

LOAD␣A

means: Clear the accumulator and add to it the contents in address A.

STO␣A

means: Store the contents of the accumulator in address A without changing the contents of the accumulator.

MPY␣A

means: Multiply the contents of the accumulator by the contents in address A and store the result in the accumulator.

SUB␣A

means: Subtract from the contents of the accumulator the contents in address, A and store the result in the accumulator.

Let us informally describe the translation through the following two examples of programs and their translations:

a)
```
READ/B,C,D,E,F,G,H
A=B*C-D*E-F-G*H
PUNCH/A
END
```

The translation is:

```
DORG/400=#DEFINE/A=#DEFINE/B=#DEFINE/C=#DEFINE/D=#DEFINE/E=#
DEFINE/F=#DEFINE/G=#DEFINE/H=#READ/B=#READ/C=#READ/D=#
READ/E=#READ/F=#READ/G=#READ/H=#LOAD/G=#MPY/H=#ST0/T+1=#
LOAD/D=#MPY/E=#ST0/T+2=#LOAD/B=#MPY/C=#ST0/T+3=#SUB/F=#
SUB/T+1=#SUB/T+2=#SUB/T+3=#ST/A=#PUNCH/A=#END=#
```

b)
```
READ/B,C,D,E,F
A=B-C*D*E-F
PUNCH/A
END
```

The translation is:

```
DORG/400=#DEFINE/A=#DEFINE/B=#DEFINE/C=#DEFINE/D=#DEFINE/E=#
DEFINE/F=#READ/B=#READ/C=#READ/D=#READ/E=#READ/F=#LOAD/C=#
MPY/D=#MPY/E=#ST0/T+1=#LOAD/B=#SUB/T+1=#SUB/F=#ST0/A=#END=#
```

A little thought will make it clear that:

a) The range language i.e. the set of sentences in the object language which are translations of sentences in the source language has got context sensitive features. One can also see that the

range language is context sensitive in a rather restricted sense in that every sentence in the range language can be broken up into four parts and the language comprising of the first part of each sentence in the range language is context free. Similarly, the languages comprising of the second, third and fourth part of each sentence in the range language are all context free.

b) In the process of translation a counting operation has to be performed. We can, however, augment a model of translation with a counter. If the computer in use is a push down machine, there is no need of a counter.

The above observations indicate that we can perhaps express each sentence in the object language as the concatenation of four sentences each of which is generated by a context-free grammar and the four context-free grammars are coupled with each other in some manner. Various questions would arise out of it. For example, suppose there is a way of coupling four context free grammars so that the above range language is generated, what will be the result if we couple more than four context-free grammars in this manner? Or to put the question in a general frame work, if n ($\geq$ 2) context free grammars are coupled with each other, will the class of languages generated be the same for all n or will the class of language be different for different n? If latter is the case, will these devices form some sort of a hierarchy?

Our interest in this thesis is to generalise SDTG's in a manner indicated in the above paragraphs and to do some investigation on the class of translation models that we will define. We

will see that one of the models of the class of translation models that we will define is able to describe the translation that we have been talking about (augmented with a counter if the computer in use is not a pushdown machine). We will formally define this translation in Appendix A).

In Chapter 1, we give the background material for our work viz. results about SDTG's and context-free languages.

In Chapter 2, a class of models for translations named n-syntax directed translation grammars (nSDTG's) is defined.

In Chapter 3, we investigate some properties of the range languages of nSDTG's.

In Chapter 4, we conclude our work by giving some conjectures and suggestions for future work.

In Appendix A, we give a grammar which describes the translation that we have been talking about in the previous paragraphs.

In Appendix B, we propose a class of machines which we feel characterise the translations defined by nSDTG's.

CHAPTER 1

SYNTAX DIRECTED TRANSLATIONS

In this chapter, we shall present some concepts and results in context free languages and syntax directed translation grammars (SDTG's). This we do to pave ground for the extensions that we shall propose in later chapters. Except for minor changes in terminology the material presented in this chapter on context free languages, has been borrowed from (5), and that on syntax directed translations from (2). The proofs of results stated can be found in the above references. In section 1.1, we give some necessary definitions and results about context free languages. In section 1.2, syntax directed translatinns are described. In section 1.3, a class of machines called pushdown assemblers which characterise syntax directed translations, are described.

1.1 Context free languages:

A context free grammar (CFG) G is defined as:

$$G = (V_N, V_T, P, S)^1,$$

where V_N and V_T are disjoint finite sets of variables or non-non-terminals and terminals, respectively. S, in V_N, is the start symbol. P is a finite set of productions of the form

---

1. We use the symbols G and P to denote grammars and productions respectively in a general sense. The type of grammar and the type of productions denoted by G and P respectively depend on the context of their use.

$A \to \alpha$, where A is in V_N and α in $(V_N \cup V_T)^*$ (we conventionally denote $V_N \cup V_T$ by V). If $A \to \alpha$ is in P, β and γ in V^*, then we say[1] $\beta A \gamma \Rightarrow \beta \alpha \gamma$ (we say $\beta A \gamma$ directly derives $\beta \alpha \gamma$ in grammar G). The relation $\overset{*}{\underset{G}{\Rightarrow}}$ is defined as follows (we say α derives β in grammar G if $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$):

(i) $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$ for every $\alpha \in V^*$

(ii) If $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$, $\beta \underset{G}{\Rightarrow} \gamma$ then $\alpha \overset{*}{\underset{G}{\Rightarrow}} \gamma$

(iii) Never is $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ unless this can be deduced from a finite number of applications of (i) and (ii).

The language generated by G denoted $L(G)$, is $\left\{w \mid w \text{ is in } V_T^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w\right\}$. $L(G)$ is called a context free language (CFL). A string of terminals and non-terminals α is called a sentential form if $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$.

A tree is a finite set of nodes connected by directed edges, which satisfy the following three conditions (if an edge is directed from node 1 to node 2, we say the edge leaves node 1 and enters node 2):

1. There is a unique node called the root, which no edge enters.

---

1. Unless otherwise stated, the following naming convention shall be used throughout the following pages:

   We shall use capital Latin alphabet letters for variables (non (non-terminals). Lower case letters at the beginning of the Latin alphabet are used for terminals. Strings of terminals are denoted by lower case letters near the end of the Latin alphabet, and strings of variables and terminals are denoted by lower case Greek letters.

2. For each node in the tree there exists a sequence of directed edges from the root to the node i.e. the tree is connected.

3. Exactly one edge enters every node except the root.

The set of all nodes n, such that there is an edge leaving a given node m and entering n, is called the set of direct descendants of m. A node n is called a descendant of node m if there is a sequence of nodes $n_1, n_2, \ldots, n_k$ such that $n_k = n$, $n_1 = m$, and for each i, n_{i+1} is a direct descendant of n_i. By convention, a node is a descendant of itself.

For each node in the tree, we can order its direct descendant Let n_1 and n_2 be direct descendants of node n, with n_1 appearing earlier in the ordering than n_2. Then we say that n_1 and all the descendants of n_1 are to the left of n_2 and all the descendants of n_2.

Let $G = (V_N, V_T, P, S)$ be a CFG. A tree is a derivation tree for G if:

1. Every node has a label, which is a symbol of V.

2. The label of the root is S.

3. If a node n has at least one descendant other than itself, and has label A, then A must be in V_N.

4. If nodes $n_1, n_2, \ldots, n_k$ are the direct descendants of node n, in order from the left, with labels $A_1, A_2, \ldots, A_k$, respectively, then $A \rightarrow A_1 A_2 \ldots A_k$ must be a production in P.

Nodes of the tree having no descendants other than themselves are called leaves of the tree.

If we read the labels of the leaves of a derivation tree from left to right, we have a sentential form. We call this string the result of the derivation tree.

A subtree of a derivation tree is a particular node of the tree together with all its descendants, the edges connecting them, and their labels.

A path in a tree is a sequence of nodes $n_1, n_2, \ldots, n_k$, such that n_{i+1} is a descendant of n_i, for $1 \leq i < k$. The length of this path is k-1. The depth of a node n is the maximum length of a path $n_1, n_2, \ldots, n_k$, such that $n_1 = n$ and n_k is a leaf. The depth of a tree is the depth of its root. If n is a node of a tree, then by definition, there must exist a unique sequence of nodes $n_1, n_2, \ldots, n_k$, such that n_1 is the root of the tree and $n_k = n$. Then the level of node n is k-1.

We now state a theorem which relates the sentential forms of the grammar with the results of its derivation trees:

Theorem 1.11:

Let $G = (V_N, V_T, P, S)$ be a context free grammar. Then for $\alpha \neq \epsilon$, $S \overset{*}{\Longrightarrow} \alpha$ if and only if there is a derivation tree in grammar G with result α.

1.2 Syntax directed translations:

A translation T from finite alphabet V_{T1} to finite alphabet V_{T2} is a subset of $V_{T1}^* \times V_{T2}^*$. (If X is any finite set of symbols, we use X^* to represent the set of finite length strings of symbols in X, including the empty string ϵ).

An element of such a translation T will be denoted (w,x), where w is in V^*_{T1} and x in V^*_{T2}. The domain D of a translation T is defined as $\{w|$ for some x in V^*_{T2}, (w, x) is in T$\}$. The range R of T is defined as $\{x|$ for some w in V^*_{T1}, (w, x) is in T$\}$. Clearly, $D \subseteq V^*_{T1}$; $R \subseteq V^*_{T2}$.

A syntax directed translation grammar (SDTG) H is defined as:

$$H = (V_N, V_{T1}, V_{T2}, R, S)^1,$$

where V_N, V_{T1} and V_{T2} are finite sets of variables (or non-terminals) input symbols and output symbols, respectively. V_N is disjoint from $V_{T1} \cup V_{T2}$. S, in V_N is the start symbol. R is a finite set of rules, a term which we shall define fully after giving the following two auxiliary definitions:

A permutation[2] π on k objects will be denoted $[i_1, i_2, \ldots, i_k]$, where i_j, for $1 \leq j \leq k$ is an integer between 1 and k, and $i_m \neq i_n$ if $m \neq n$. $\pi(j)$ is defined to be i_j; $\hat{\pi}(j)$ is that k such that $\pi(k) = j$.

A translation form of H is a triple (α, β, π) where α is in $(V_N \cup V_{T1})^*$, β is in $(V_N \cup V_{T2})^*$ and π is a permutation. The number of variables in α and β must be equal, say k in each and the number of appearances of each variable in α and β should be the

---

1. We use the symbols H and R to denote translation grammars and rules respectively in a general sense. The type of translation grammar that H denotes or the type of rules that comprise R will be clear by the context in which these symbols are used.

2. We shall be using this term in later chapters also.

same. π must then be a permutation on k objects. For all i, $1 \leq i \leq k$, the i-th variable of α (from the left) is the same as the $\pi(i)$-th variable of β (from the left). We say that the i-th variable of α and the $\pi(i)$-th variable of β correspond.[1]

A rule is an object $A \rightarrow (\alpha, \beta, \pi)$, where A is a variable and (α, β, π) is a translation form.

Suppose $(\alpha_1, \beta_1, \pi_1)$ is a translation form of H and A the i-th variable of α_1 from the left. Also, suppose $A \rightarrow (\gamma, \delta, \pi)$ is a rule. Then we can construct a translation form $(\alpha_2, \beta_2, \pi_2)$ by replacing the i-th variable of α_1 by γ to obtain α_2 and the $\pi_1(i)$-th variable of β_1 by δ to obtain β_2. π_2 is the permutation such that the variables of α_1 other than the i-th correspond to the same symbol in β_2 as in β_1 and each variable of γ corresponds to the variable of δ to which it corresponded according to π. We show that π_2 can always be constructed, as follows:

Let have m variables, $m \geq 0$. π_2 is defined by:

(i) For all $j < i$, if $\pi_1(j) < \pi_1(i)$, then $\pi_2(j) = \pi_1(j)$, and if $\pi_1(j) > \pi_1(i)$, then $\pi_2(j) = \pi_1(j) + m-1$.

(ii) For all $j > i$, if $\pi_1(j) < \pi_1(i)$, then $\pi_2(j+m-1) = \pi_1(j)$, and if $\pi_1(j) > \pi_1(i)$, then $\pi_2(j+m-1) = \pi_1(j)+m-1$.

(iii) For all d, $1 \leq d \leq m$, $\pi_2(i + d - 1) = \pi_1(i) + \pi(d) - 1$.

---

1. If it is obvious which variables of α and β correspond, because no variables appear more than once in α or β, we shall often omit the permutation and write the translation form as (α, β). If there are no variables in α, β, there is obviously no need of having the permutation present in the translation form.

We say, if all of the above is true, that $(\alpha_1, \beta_1, \pi_1) \underset{H}{\Longrightarrow} (\alpha_2, \beta_2, \pi_2)$. (we say that $(\alpha_1, \beta_1, \pi_1)$ directly derives $(\alpha_2, \beta_2, \pi_2)$ in grammar H).

Example 1.21:

Suppose $(\alpha_1, \beta_1, \pi_1)$ is a translation form of some SDTG H. Let α_1 = aABbBba, β_1 = BccBAd and π_1 = [3, 1, 2]. Note that the last B of α_1 is identified with the last B of β_1 since $\pi_1(3) = 2$. Let B → (aAA, cAAd, [2, 1]) be a rule of H. We can apply this rule to the last B of α_1 and the last B of β_1 to show that $(\alpha_1, \beta_1, \pi_1) \underset{H}{\Longrightarrow} (\alpha_2, \beta_2, \pi_2)$ where α_2 = aABbaAAba, β_2 = BcccAAdAd, and π_2 = [4,1,3,2].

We define the relation $\overset{*}{\underset{H}{\Longrightarrow}}$ by:

(i) $(\alpha, \beta, \pi) \overset{*}{\underset{H}{\Longrightarrow}} (\alpha,\beta,\pi)$, for every $\alpha \in (V_N \cup V_{T1})^*$ and $\beta \in (V_N \cup V_{T2})^*$

(ii) If $(\alpha_1, \beta_1, \pi_1) \overset{*}{\underset{H}{\Longrightarrow}} (\alpha_2, \beta_2, \pi_2)$ and $(\alpha_2, \beta_2, \pi_2) \underset{H}{\Longrightarrow} (\alpha_3, \beta_3, \pi_3)$ then $(\alpha_1, \beta_1, \pi_1) \overset{*}{\underset{H}{\Longrightarrow}} (\alpha_3, \beta_3, \pi_3)$.

(We say that $(\alpha_1, \beta_1, \pi_1)$ derives $(\alpha_3, \beta_3, \pi_3)$ in grammar H)

The syntax directed translation (SDT) defined by H, denoted T(H) is

$$\left\{(x, y) \mid (S, S) \overset{*}{\underset{H}{\Longrightarrow}} (x, y) \text{ and } x \in V_{T1}^*, y \in V_{T2}^*\right\}$$ [1]

---

1. Note that permutations have been omitted from the translation forms related by $\overset{*}{\underset{H}{\Longrightarrow}}$ since obviously, we will not get any additional information by having them present in these translation forms. We will always omit permutations if it does not result in any loss of information.

We now state a result which characterises the domain and range of any syntax directed translation.

Lemma 1.21:

The domain and range of any syntax directed translation are CFL's.

It has been shown that the domain of T such that $T = T(H)$ for some SDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$, is generated by the CFG $G_i = (V_N, V_{T1}, P, S)$, where

$$P = \{A \to \alpha \mid \text{for some } \beta \text{ and } \pi,\ A \to (\alpha, \beta, \pi) \text{ is in } R\}.$$

G_i is said to be the input grammar of H. Similarly the range of T has been shown to be generated by the CFG $G_o = (V_N, V_{T2}, P, S)$, where $P = \{A \to \beta \mid \text{for some } \alpha \text{ and } \pi,\ A \to (\alpha, \beta, \pi) \text{ is in } R\}$.

We now define a restricted type of SDTG called simple syntax directed translation grammar (SSDTG).

If $H = (V_N, V_{T1}, V_{T2}, R, S)$ is an SDTG such that for all $A \to (\alpha, \beta, \pi)$ in R, π is always an identity permutation, then H is said to be simple syntax directed translation grammar (SSDTG) and T(H) is called a simple syntax directed translation (SSDT). Obviously, in an SSDTG, there is no reordering of the variables appearing on the right hand side of rules.[1] The following theorem establishes the relationship between SDT's and SSDT's.

Theorem 1.21:

SSDT's are a proper subset of the SDT's.

---

1. While dealing with SSDTG's we shall invariably omit the permutation and write the translation form as (α, β).

It has been shown that the translation T(H) defined by the SDTG:

$$H = (\{S, A\}, \{a, b, c\}, \{a, b, c\}, R, S),$$

where R consists of the rules:

$S \rightarrow (AcA, AcA, [2, 1])$

$A \rightarrow (aA, aA)$

$A \rightarrow (bA, bA)$

$A \rightarrow (a, a)$

$A \rightarrow (b, b)$

cannot be defined by any SSDTG. On the other hand, an SSDTG is by definition also an SDTG.

We shall now define the <u>order</u> of SDT's. Based on this definition an infinite hierarchy of SDT's has been established.

An SDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$ is said to be of <u>order</u> k if for all rules $A \rightarrow (\alpha, \beta, \pi)$ in R, there are no more than k variables in α. An SDT T is of order k if $T = T(H)$ for some SDTG H of order k.

The hierarchy results are as follows: ($\mathcal{T}_k$ is set of SDT's of orde

<u>Theorem 1.22</u>:

$\mathcal{T}_2$ properly contains $\mathcal{T}_1$.

<u>Theorem 1.23</u>:

$\mathcal{T}_2 = \mathcal{T}_3$

<u>Theorem 1.24</u>:

For each $k \geq 4$, $\mathcal{T}_k$ properly contains $\mathcal{T}_{k-1}$.

It has been shown that the set of translation T_k, $k \geq 4$, defined as follows, are not of order k-1.

Let i be an integer greater than 1. Define π_{2i} to be the permutation $[i+1, 1, i+2, 2, \ldots, 2i, i]$. That is, $\pi_4 = [3,1,4,2]$, $\pi_6 = [4,1,5,2,6,3]$, etc. Define π_{2i-1} to be the permutation $[i, 2i-1, 1, 2i-2, 2, \ldots, i+1, i-1]$. That is, $\pi_5 = [3,5,1,4,2]$, $\pi_7 = [4,7,1,6,2,5,3]$, etc. Let the input as well as the output symbols of the translation T_k be $\{a_1, a_2, \ldots, a_k\}$.

Then T_k is defined to be the translation:

$$\{(a_1^{i_1} a_2^{i_2} \ldots a_k^{i_k}, b_1^{j_1} b_2^{j_2} \ldots b_k^{j_k}) \mid \text{for } 1 \leq m \leq k, i_m \geq 1, b_{\pi(m)} = a_m \text{ and } j_{\pi(m)} = i_m\}.$$

1.3 Pushdown assemblers:

In this section, we shall describe two classes of machines called Type A pushdown assemblers (APA) and type B pushdown assemblers (BPA). Results have been established which relate these machines with the SDT's. As we shall see, pushdown assemblers are pushdown automata (PDA) with additional features. To start with, we shall therefore give a qualitative description of PDA.

A pushdown automaton (PDA) is an abstract machine consisting of a finite state control attached to a pushdown list. The finite control has an input terminal at which input symbols appear when requested by the finite control. The pushdown list can contain any finite length string of tape symbols. The finite control can

read the top symbol of the list, and in any move can replace the top symbol by a finite length string of symbols, including the empty string. The device is nondeterministic and may have any finite number of choices in each situation. A sequence of input symbols is accepted if some choice of moves of the PDA using that sequence of inputs causes the pushdown list to become empty or causes the finite control to enter a prespecified state called the final state of the machine. The language accepted by PDA is a context free language, and every context free language is accepted by a PDA.

We shall now describe the type A pushdown assemblers (APA's). Here, associated with each tape symbol on the pushdown list are k passive registers. A tape symbol together with its registers is called a 'level'. Each register can be empty or hold a string of output symbols. Such a situation is shown in Figure 1.31, where k = 2 and the pushdown list contains the tape symbols CBA, with C on the top of the list. Associated with A are two registers , the empty (∅ indicates an empty register) and the second holding string w. Both registers associated with B are empty. The first register associated with C contains string x while the second register is empty.

C	x	∅
B	∅	∅
A	∅	w

Figure 1.31

Suppose C, at the top of the list, is replaced by string ED. Symbol D would replace C in Figure 1.31, and E would appear above D with empty registers. The result appears in Figure 1.32.

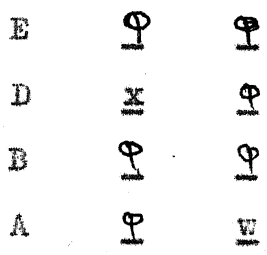


Figure 1.32

The APA can write a finite string in any empty register at the highest level. For example, it could write y and z in the first and second registers of the top level, leaving the situation of Figure 1.33.

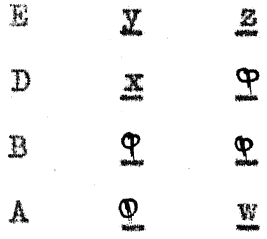


Figure 1.33

If the top tape symbol is erased, the contents of its registers are concatenated in order. If a register is empty, it is treated as though it contained ϵ (the empty string). The resulting string then 'waits' at the top of the list to be placed, on the next move, in an empty register. If E in Figure 1.33 were erased, yz would be passed down to the level below and would appear temporarily to the left of D as in Figure 1.34.

yz	D	x	Φ
	B	Φ	Φ
	A	Φ	w

Figure 1.34

Next, yz is placed in an empty register at the top level. In this case, it must be the second register. If the APA tries to write into a register which is not empty, it 'jams' and can make no further moves. Figure 1.34 thus should become Figure 1.35.

D	x	yz
B	Φ	Φ
A	Φ	w

Figure 1.35

If next, D were erased and the string resulting from the concentration of its registers were stored in the second register of the next lower level, the list would be as in Figure 1.36.

B	Φ	xyz
A	Φ	w

Figure 1.36

Finally, suppose that on successive moves, B is erased. The resulting string, xyzw, would have no place to go, and will be deemed output of the APA; xyzw would be deemed a translation of whatever input string causes the sequence of moves, the terminal portion of which we have been describing.

We shall now formally describe a k-register type A pushdown assembler (kAPA):

A kAPA M is defined as:

$$M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0),^{1}$$

where Q, Σ, Δ, and Γ are finite sets of states, input symbols, output symbols and tape symbols, respectively. q_0, in Q, is the start state. Z_0, in Γ, is the start symbol. λ, μ and ν are mappings which indicate the allowable moves of M. λ controls changes of the tape symbols (not register contents) on the pushdown list. μ controls the entry of finite length output strings into registers. ν controls the insertion into registers of output strings which have been displaced temporarily by the erasure of the top symbol on the pushdown list (as in Figure 1.34).

λ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times \Gamma^*$.

If $\lambda(p, a, Z)$ contains (q, ϵ), then, if the topmost tape symbol is Z, M can erase the top level, transfer from state p to q and use input a (possibly ϵ). If $\lambda(p, a, Z)$ contains $(q, X_1 X_2 \ldots X_m)$[2], $m \geq 1$, M can replace Z by X_m at the top level, then grow new levels on the top of the pushdown list with empty registers and tape symbols $X_1, X_2, \ldots, X_{m-1}$, in order from the top.

---

1. The symbols M, λ, μ, ν are used in general sense. What they specifically denote will be clear by the context of their use.
2. While talking about machines, we shall be using capital Latin alphabet letters for tape symbols.

μ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times \Delta^* \times \{1, 2, \ldots, k\}$.

If $\mu(p, a, Z)$ contains (q, x, i), then when Z is the top tape symbol, M can use input a, transfer from state p to q and write output string x in the i-th register of the top level, provided that register is empty.

ν is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the subsets of $Q \times \{1, 2, \ldots, k\}$.

If $\nu(p, a, Z)$ contains (q, i), then if Z is the top tape symbol and the level above has just been erased, M can use input a, transfer from state p to q and store the data passed from the level above in the i-th register of the current top level. Again, that register must be empty.

A <u>configuration</u> of M is denoted (q, α), where q is in Q, and α is a string either of the form $Z_1 t_1 Z_2 t_2 \ldots Z_m t_m$ or $[w]\ Z_1 t_1 Z_2 t_2 \ldots Z_m t_m$, where $Z_1, Z_2, \ldots, Z_m$ are in Γ, $t_1, t_2, \ldots, t_m$ are k-tuples of elements in $\Delta^* \cup \{\varphi\}$ and w is in Δ^*. t_i, $1 \leq i \leq m$, represents the contents of the k registers associated with Z_i. φ denotes an empty register. The string α is prefixed by $[w]$ if a tape symbol has been erased on the previous move, a condition akin to that of Figure 1.34[1].

---

1. When dealing with a k-register type A machine (kAPA), t_o will stand for the k-tuple $(\varphi, \varphi, \ldots, \varphi)$. When dealing with a k-register type B machine (kBPA), which we shall discuss later, t_o will stand for the k-tuple $(\epsilon, \epsilon, \ldots, \epsilon)$. There is a need to differentiate between empty register and register containing empty string because of the requirement that we can write in a register only when it is empty i.e. contains no string, in case of kAPA.

For any q in Q, a in $\Sigma \cup \{\epsilon\}$, Z in Γ and k-tuble t, suppose $\lambda(q, a, Z)$ contains $(p, Z_1, Z_2, \ldots, Z_m)$, $m > 1$, then we may write, $a:(q, Zt\alpha) \vdash_M (p, Z_1 t_o Z_2 t_o \ldots Z_{m-1} t_o Z_m t\alpha)$. If $m = 1$, then we may write,

$$a : (q, Zt\alpha) \vdash_M (p, Z_1 t\alpha).$$

If $\lambda(q, a, Z)$ contains (p, ϵ), then we may write,

$$a : (q, Z(w_1, w_2, \ldots, w_k)\ \alpha) \vdash_M (p, [w_1 w_2 \ldots w_k]\alpha).$$

Here $w_1 w_2 \ldots w_k$ is the concatenation of $w_1, w_2, \ldots, w_k$, with φ taken to be ϵ.

Suppose $\nu(q,a,Z)$ contains (p, i), then we may write,

$$a : (q, [w]\ Z(x_1, x_2, \ldots, x_k)\ \alpha)$$

$$\vdash_M (p, Z(x_1, \ldots, x_{i-1}, w, x_{i+1}, \ldots, w_k)\ \alpha),$$

provided $x_i = \varphi$. Finally, if $\mu(q, a, Z)$ contains (p, w, i), then we may write,

$$a : (q, Z(x_1, x_2, \ldots, x_k)\ \alpha) \vdash_M (p, Z(x_1, \ldots, x_{i-1}, w, x_{i+1}, \ldots,$$

again provided $x_i = \varphi$.

The symbol $\vdash_M^*$ is defined as follows:

For any configuration (q, α),

$\epsilon : (q, \alpha) \vdash_M^* (q, \alpha)$. For w in Σ^* and a in $\Sigma \cup \{\epsilon\}$, if $w : (q_1, \alpha_1) \vdash_M^* (q_2, \alpha_2)$ and $a: (q_2, \alpha_2) \vdash_M (q_3, \alpha_3)$, then $wa : (q_1, \alpha_1) \vdash_M^* (q_3, \alpha_3)$.

The <u>translation defined by</u> M, denoted $\tau(M)$, is $\{(w, x) \mid w: (q_o, Z_o t_o) \vdash_M^* (q, [x])$ for some q in $Q\}$.

The type B pushdown assembler (BPA) differs from the type A pushdown assembler only in the way output strings can be stored in registers. The BPA always concatenates strings to the left or right of the present contents of a register. The BPA is thus a allowed to write into a nonempty register. This modification extends the range of translations definable by a pushdown assembler.

Formally, a k-register type B pushdown assembler (kBPA) M is defined as:

$$M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0),$$

where all symbols (within parenthesis) have the same meaning as for the APA except:

(i) μ maps $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times \Delta^* \times \{1,2,\ldots, k\} \times \{L,R\}$. If $\mu(p,a,Z)$ contains (q,x,i,D), then M can concatenate x to the left or right end of the contents of the i-th register at the top level depending on whether D = L or R, respectively.

(ii) ν maps $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the subsets of $Q \times \{1, 2, \ldots, k\} \times \{L, R\}$. If $\nu(p, a, Z)$ contains (q, i, D), then M can concatenate data passed down from the level above to the left or right of the current contents of register i of the top level, depending on whether D = L or R, respectively.

A configuration of M is denoted (q, α), as for the APA. The k-tuples in string α are composed of elements in Δ^*, rather than $\Delta^* \cup \{\varphi\}$. For a BPA ϵ, instead of ϕ, can be used for an empty register[1].

---

1. As noted earlier, in case of BPA, t_0 will denote the k-tuple $(\epsilon, \epsilon, \ldots, \epsilon)$.

The symbol $\vdash_M$ is defined by:

(i) If $\lambda(q, a, Z)$ contains $(p, Z_1Z_2 \ldots Z_m)$, $m > 1$, then
$a : (q, Zt\alpha) \vdash_M (p, Z_1t_oZ_2t_o \ldots Z_{m-1}t_oZ_mt\alpha)$,

(ii) If $\lambda(q, a, Z)$ contains (p, Y), then
$a : (q, Zt\alpha) \vdash_M (p, Yt\alpha)$.

(iii) If $\lambda(q, a, Z)$ contains (p, ϵ), then
$a : (q, Z(w_1, w_2, \ldots, w_k)\ \alpha) \vdash_M (p, [w_1w_2 \ldots w_k]\ \alpha)$.

(iv) If $\nu(q, a, Z)$ contains (p, i, D), then
$a : (q, [w]\ Z(x_1, x_2, \ldots, x_k)\ \alpha) \vdash_M (p, Z(y_1, y_2, \ldots, y_k)\ \alpha)$,
where $y_j = x_j$ for $j \neq i$, $y_i = wx_i$ or x_iw, as $D = L$ or R.

(v) If $\mu(q, a, Z)$ contains (p, w, i, D), then
$a : (q, Z(x_1, x_2, \ldots, x_k)\ \alpha) \vdash_M (p, Z(y_1, y_2, \ldots, y_k)\ \alpha)$,
where $y_j = x_j$ for $j \neq i$ and $y_i = wx_i$ or x_iw as $D = L$ or R.

The symbol $\vdash_M^*$ is defined from $\vdash_M$ exactly as for the APA. The <u>translation defined by</u> M, denoted $\mathcal{C}(M)$, is $\{(w,x) \mid w : (q_o, Z_ot_o) \vdash_M^* (q, [x])$ for some q in $Q\}$. If M is a k-register BPA, then $\mathcal{C}(M)$ will be called a <u>B-translation of order k</u>.

We shall now state results which relate the pushdown assemblers with the syntax directed translations.

<u>Theorem 1.31:</u>

If a translation T is an SDT of order $k \geq 2$, then $T = \mathcal{C}(M)$ for some kAPA M.

Theorem 1.32:

If $T = \mathcal{T}(M)$ for a k APA $M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \sqrt{}, q_0, Z_0$ then $T = T(H)$ for an SDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$ of order k.

Theorem 1.33:

If T is an SDT of order 2k, then T is defined by a kBPA M.

Theorem 1.34:

For $k \geq 2$, there exist B-translations of order k which are not SDT's of any order.

Theorem 1.35:

If T is a B-translation of order 1, then T is defined by a 2-register APA.

Theorem 1.36:

A translation T is a simple syntax directed translation (SSDT) if and only if T can be defined by a pushdown transducer.

Theorem 1.37:

A translation T is definable by a 1-register right concatenating BPA if and only if T is an SSDT.

CHAPTER 2

n - SYNTAX DIRECTED TRANSLATIONS

In this chapter, we shall propose a natural generalisation of the concept of syntax directed translation grammars discussed in the previous chapter. We shall also generalise the concept of context free languages so as to characterise the range language of n-syntax directed translations. In section 2.1, we define n-syntax directed translations. In section 2.2, we define n-context free languages and show that the range languages of n-syntax directed translations are the same as n-context free languages.

2.1 n-Syntax directed translations:

An n-syntax directed translation grammar (nSDTG) H is defined as:

$$H = (V_N, V_{T1}, V_{T2}, R, S),$$

where V_N, V_{T1}, V_{T2} are finite sets of non-terminals (Variables), input symbols, and output symbols, respectively. V_N is disjoint from $V_{T1} \cup V_{T2}$. S, in V_N, is the start symbol. R is a finite set of n-rules; we shall define this term fully after giving the following definition:

An n-translation form of H is:

$$(\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)),$$

where, α is in $(V_N \cup V_{T1})^*$, β_i, $1 \leq i \leq n$, is in $(V_N \cup V_{T2})^*$, $\beta_1\beta_2 \cdots \beta_n \neq \epsilon$ and π_i, $1 \leq i \leq n$, is a permutation.[1] The number of variables in α and β_i must be equal, say k in each and the number of appearances of each variable in α and β_i should be the same. π_i must then be a permutation on k objects. For all i and j, $1 \leq i \leq k$, $1 \leq j \leq n$, the i-th variable of α (from the left) is the same as the $\pi_j(i)$-th variable of β_j (from the left). We say that the i-th variable of α and the $\pi_j(i)$-th variable of β_j <u>correspond</u>[2].

An <u>n-rule</u> is defined as:

$$A \rightarrow (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$$

where A is a variable and $(\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is an n-translation form.

Suppose $(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$ is an n-translation form of H and A the i-th variable of α_1 from the left. Also, suppose $A \rightarrow (\gamma, (\delta_1, \pi_1), \ldots, (\delta_n, \pi_n))$ is an n-rule. Then we can construct an n-translation form replacing the i-th variable of α_1 by γ to obtain α_2 and the $\pi_{1j}(i)$-th variable of β_{1j} by δ_j to obtain β_{2j}, $1 \leq j \leq n$. π_{2j} is the permutation such that the variables of α_1 other than the i-th correspond to the same symbol in β_{2j} as in β_{1j} and each variable of γ corresponds to the variable of δ_j to which it corresponded according to π_j.

---

1. The term permutation has been defined in the previous chapter.

2. If it is obvious which variables of α and β_j correspond with each other because no variable appears more than once in α or β_j we shall often omit the permutations from the n-translation forms.

Formally, let γ have m variables, $m \geq 0$. π_{2j} is defined by:

(i) For all $k < i$ if $\pi_{1j}(k) < \pi_{1j}(i)$, then $\pi_{2j}(k) = \pi_{1j}(k)$, and if $\pi_{1j}(k) > \pi_{1j}(i)$, then $\pi_{2j}(k) = \pi_{1j}(k+m-1)$.

(ii) For all $k > i$, if $\pi_{1j}(k) < \pi_{1j}(i)$, then $\pi_{2j}(k+m-1) = \pi_{1j}(k)$, and if $\pi_{1j}(k) > \pi_{1j}(i)$, then $\pi_{2j}(k+m-1) = \pi_{1j}(k)+m-1$.

(iii) For all d, $1 \leq d \leq m$, $\pi_{2j}(i+d-1) = \pi_{1j}(i) + \pi_j(d)-1$.

If all the above is true, we say that,

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{H}{\Rightarrow} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}$$

We define the relation $\overset{*}{\underset{H}{\Rightarrow}}$ as follows:

(i) $(\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)) \overset{*}{\underset{H}{\Rightarrow}} (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$, for every $\alpha \in (V_N \cup V_{T1})^*$ and β_j, $1 \leq j \leq n$, $\in (V_N \cup V_{T2})^*$.

(ii) If $(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{H}{\Rightarrow}} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

and $(\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n})) \underset{H}{\Rightarrow} (\alpha_3, (\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$,

then $(\alpha_1, (\beta_{11}\ \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{H}{\Rightarrow}} (\alpha_3, (\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$.

The <u>n-syntax directed translation</u> (nSDT) defined by H, denoted T(H) is:

$$\left\{(x, y) \mid (S,S,\ldots,S)_n^1 \overset{*}{\underset{H}{\Rightarrow}} (x, y_1, y_2, \ldots, y_n) \text{ and } y = y_1 y_2 \ldots y_n\right\}$$

1. Unless otherwise indicated we shall throughout use the notation $(A,A,\ldots,A)_n$ to represent the n-tuple $(A,A,\ldots,A)$.

Example 2.11:

We shall give an example of 2SDT (nSDT with n = 2).

Let H = ({S,A}, {a,b,c},{a,b,c}, R,S)

be a 2SDTG where R consists of the 2-rules:

1. S → (AcAc,(AcAc, [2,1]), (AcAc, [2, 1]),
2. A → (aA,aA,aA),
3. A → (bA,bA,bA),
4. A → (a,a,a), and
5. A → (b,b,b).

By the first rule in R

$$(S,S,S) \underset{H}{\Longrightarrow} (AcAc, (AcAc, [2,1]), (AcAc, [2,1]))$$

Now we can apply rules 2,3,4,5 to replace to variable A. Note, however, that while the use of rules 4 or 5 replaces A by a symbols a or b respectively, use of rules 2 or 3 replace A's by the strings aA's or bA's respectively which contain exactly one A. Thus, the only sequence in which the rules can be applied is applying rule 1 first, then applying rules 2 or 3 any number of times (including not applying them at all) and finally applying rules 4 or 5 as many times as in necessary. It is easy to see that this way

$$(S,S,S) \overset{*}{\underset{H}{\Longrightarrow}} (wcxc, xcwc, xcwc),$$

where $w, x \in (a, b)^*$, and T(H) contains all pairs (wcxc, xcwcxcwc). That these are the only pairs in T(H) is clear from the above argument.

If $T = T(H)$ for some nSDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$, then the domain of T is CFL. For example, the domain of T is generated by the CFG $G = (V_N, V_{T1}, P, S)$, where $P = \{A \to \alpha \mid$ for some $\beta_1, \pi_1, \ldots, \beta_n, \pi_n$, $A \to (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is in $R\}$. As the above example shows, the range of T, in general, is not CFL. In the next section we shall generalise the concept of context free languages, so as to characterise the range of T.

If $H = (V_N, V_{T1}, V_{T2}, R, S)$ is an nSDTG such that for all $A \to (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ in R, π_i, $1 \leq i \leq n$, is always an identity permutation, then H is said to be <u>simple</u> and T(H) is called <u>n-simple syntax directed translation</u> (nSSDT).

An nSDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$ is said to be of <u>order</u> k if for all rules $A \to (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ in R, there are no more than k variables in α. An nSDT T is of <u>order</u> k if $T = T(H)$ for some nSDTG of order k.

2.2 <u>n-Context free languages</u>:

An <u>n-context free grammar</u> (nCFG) G is defined as:

$$G = (V_N, V_T, P, S),$$

where V_N and V_T are disjoint finite sets of <u>non-terminals (variables)</u> and <u>terminals</u> respectively ($V = V_N \cup V_T$). S, in V_N, is the <u>start symbol</u>. P is a finite set of <u>n-productions</u>. Before defining this term fully, we shall give the following definitions:

An <u>n-form</u> of G is an n-tuple:

$$((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)),$$

where β_i, $1 \leq i \leq n$, is in V^*, $\beta_1\beta_2 \ldots \beta_n \neq \epsilon$ and π_i, $1 \leq i \leq n$, is a permutation. The number of variables in β_i's must be equal, say k in each and the number of appearances of each variable in them should be the same. π_i is a permutation on k objects. For all $1 \leq i \leq k$, the $\pi_j(i)$-th variables of β_j (from the left), $1 \leq j \leq n$, are the same. We say that the $\pi_j(i)$-th variables of β_j correspond with each other. (β_i, π_i) is called the i-th expansion of the n-form.

An n-production is defined as:

$$A \rightarrow ((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)),$$

where A is a variable and $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is an n-form. The n-form $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is called the output expansion of the n-production.

Similarly as for nSDT's, we define here the relation $\underset{G}{\Longrightarrow}$ as follows:

Let, $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$ be an n-form of G and A the $\pi_{1j}(i)$-th variable of β_{1j}, $1 \leq j \leq n$, from the left. Also, suppose $A \rightarrow ((\gamma_1, \pi_1), \ldots, (\gamma_n, \pi_n))$ is an n-production, γ_i having m variables. Then,

$$((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{G}{\Longrightarrow} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

(we say that $(\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})$ directly n-derives $(\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n})$ in grammar G by expansion of the variable A) if and only if the following is true:

(i) β_{2j} is obtained from β_{1j} by replacing the $\pi_{1j}(i)$-th variable (i.e. A) by γ_j.

(ii) For all $k < i$, if $\pi_{1j}(k) < \pi_{1j}(i)$, then $\pi_{2j}(k) = \pi_{1j}(k)$, and if $\pi_{1j}(k) > \pi_{1j}(i)$, then $\pi_{2j}(k) = \pi_{1j}(k) + m-1$.

(iii) For all $k > i$, if $\pi_{1j}(k) < \pi_{1j}(i)$, then $\pi_{2j}(k+m-1) = \pi_{1j}(k)$, and if $\pi_{1j}(k) > \pi_{1j}(i)$, then $\pi_{2j}(k+m-1) = \pi_{1j}(k)+m-1$.

(iv) For all d, $1 \leq d \leq m$, $\pi_{2j}(i+d-1) = \pi_{1j}(i) + \pi(d)-1$.

We shall illustrate the direct n-derivation ($\underset{G}{\Longrightarrow}$) in the grammar through an example:

Example 2.21:

Let G = ({ S, A, B} , {a, b, c} , P, S) be a 2 CFG where P consists of the 2-productions:

1. S → ((aAaBBcA, [2,1,3,4]), (AABB, [4,2,3,1])),
2. A → ((aAaBbBc, [1,2,3]), (bBcBaA, [3,1,2])),
3. B → ((ABB, [2,3,1]), (aaBAB, [3,1,2])),
4. A → (ϵ, ϵ), and
5. B → (abc, ϵ).

Now, ((aaAAAbB, [2,3,1,4]), (cBAccAA, [2,4,3,1])) is a 2-form in G. Let us apply the 2-production no. 3 to this 2-form.

Using the procedure given above, we will get the 2-form:

((aaAAAbABB, [2,3,1,5,6,4]), (caaBABccAA, [4,6,5,3,1,2])).

We can therefore write:

((aaAAAbB, [2,3,1,4]), (cBAccAA, [2,4,3,1]))

$\underset{G}{\Longrightarrow}$ (aaAAAbABB, [2,3,1,5,6,4]),(caaBABAccAA,[4,6,5,3,1,

The relation $\overset{*}{\underset{G}{\Longrightarrow}}$ is defined as follows:

(i) $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)) \overset{*}{\underset{G}{\Longrightarrow}} ((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$

for every β_j, $1 \leq j \leq n$, $\in V^*$.

(ii) If $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$

$$\overset{*}{\underset{G}{\Longrightarrow}} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

and $((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

$$\underset{G}{\Longrightarrow} ((\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$$

then $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$

$$\overset{*}{\underset{G}{\Longrightarrow}} ((\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$$

(we say that $(\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})$ <u>n-derives</u> $((\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$ in grammar G).

The n-tuple $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is called an <u>n-sentential form</u> if β_j, $1 \leq j \leq n$, $\in V^*$ and

$$(S, S, \ldots, S)_n \overset{*}{\underset{G}{\Longrightarrow}} ((\beta_1, \pi_1), (\beta_2, \pi_2), \ldots, (\beta_n, \pi_n)).$$

Also if $((\beta_1, \pi_1), (\beta_2, \pi_2), \ldots, (\beta_n, \pi_n))$ is an n-sentential form in grammar G, β_i is called the <u>i-th component of</u> the n-sentential form.

We define the <u>language generated</u> by G denoted L(G) to be:

$$\{w \mid w \text{ is in } V_T^*,\ (S, S, \ldots, S)_n \overset{*}{\underset{G}{\Longrightarrow}} (w_1, w_2, \ldots, w_n),\ \text{and}\ w = w_1 w_2 \ldots w_n\}.$$

If $G = (V_N, V_T, P, S)$ is an nCFG such that for all $A \rightarrow ((\beta_1, \pi_1), (\beta_2, \pi_2), \ldots, (\beta_n, \pi_n))$ in P, π_i, $1 \leq i \leq n$, is always an identity permutation, then G is said to be simple and L(G) is called n-simple context free language (nSCFL).

Now we shall give some examples to clarify the concepts:

Example 2.22:

In example 2.21, we saw that if $G = (\{S,A,B\}, \{a,b,c\}, P, S)$ is a 2 CFG, P consisting of:

1. $S \rightarrow ((aAaBBcA, [2,1,3,4]), (AABB, [4,2,3,1]))$,
2. $A \rightarrow ((aAaBbBc, [1,2,3]), (bBcBaA, [3,1,2]))$,
3. $B \rightarrow ((ABB, [2,3,1]), (aaBAB, [3,1,2])$,
4. $A \rightarrow (\epsilon, \epsilon)$, and
5. $B \rightarrow (abc, \epsilon)$,

then, $((aaAAAbB, [2,3,1,4]), (cBAccAA, [2,4,3,1]))$

$$\underset{G}{\Rightarrow} ((aaAAAbABB, [2,3,1,5,6,4]), (caaBABAccAA, [4,6,5,3,1,2])),$$

From the definition of the relation $\overset{*}{\underset{G}{\Rightarrow}}$ it follows that,

$((aaAAAbB,[2,3,1,4]),(cBAccAA,[2,4,3,1]))$

$$\overset{*}{\underset{G}{\Rightarrow}} ((aaAAAbABB,[2,3,1,5,6,4]),(caaBABAccAA,[4,6,5,3,1,2])).$$

If we apply the 2-production no. 2 to expand A, the 2nd variable (from left) of the first expansion of the 2-form:

$((aaAAAbABB,[2,3,1,5,6,4]),(caaBABAccAA,[4,6,5,3,1,2])),$

We have;

$$((aaAAAbABB,[2,3,1,5,6,4]),(caaBABAccAA,[4,6,5,3,1,2]))$$

$$\underset{G}{\Longrightarrow} ((aaAaAaBbBcAbABB,[2,3,4,5,1,7,8,6]),$$

$$((caaBABbBcBaAccAA,\ [6,4,5,8,7,3,1,2])).$$

Again, from the definition of $\overset{*}{\underset{G}{\Longrightarrow}}$, it follows that,

$$((aaAAAbB,[2,3,1,4]),\ (cBAccAA,[2,4,3,1]))$$

$$\overset{*}{\underset{G}{\Longrightarrow}} ((aaAaAaBbBcAbABB,\ [2,3,4,5,1,7,8,6]),$$

$$((caaBABbBcBaAccAA,\ [6,4,5,8,7,3,1,2])).$$

Example 2.23:

Let H = ($\{S\}$, $\{a,b,c\}$, P, S) be a 2SCFG (2-simple context free grammar), where P consists of:

1. $S \rightarrow (aSb,\ cS)^1$
2. $S \rightarrow (ab,c)^1$

By applying the first 2-production n-1 times followed by an application of the second 2-production, we have:

$$(S,S) \underset{G}{\Longrightarrow} (aSb,\ cS) \underset{G}{\Longrightarrow} (aaSbb,\ ccS) \underset{G}{\Longrightarrow} \ \ldots$$

$$\ldots \underset{G}{\Longrightarrow} (a^{n-1}\ Sb^{n-1},\ c^{n-1}\ S) \underset{G}{\Longrightarrow} (a^nb^n, c^n).$$

Therefore, $\{a^nb^nc^n \mid_{n\geq 1}\} \subseteq L(G)$.

---

1. Note that we have deleted permutations from the 2-forms in this case. This is in keeping with the convention for nSCFG's.

It can be easily shown that $L(G) \subseteq \{a^n b^n c^n \mid_{n \geq 1}\}$ and therefore $L(G) = \{a^n b^n c^n \mid_{n \geq 1}\}$.

We shall now show that the range language of an n-syntax directed translation is the same as an n-context free language.

Theorem 2.21:

The range language of an n-SDT is an n-CFL.

Proof:

Let $H = (V_N, V_{T1}, V_{T2}, R, S)$ be an nSDTG. We construct from this grammar an nCFG as follows:

Let $G = (V_N, V_{T2}, P, S)$,

where, $P = \{A \to ((\beta_1,\pi_1),\ldots,(\beta_n,\pi_n)) \mid$

$A \to (\alpha, (\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is in R

for some $\alpha\}$.

However, reflexive n-productions, should any appear are deleted from P.

Let x be the range language of H. Thus, for some w, $(w, x_1, x_2, \ldots, x_n)$ is in T(H) and $x = x_1 x_2 \ldots x_n$. Therefore,

$$(S,S, \ldots, S)_n \overset{*}{\underset{H}{\Longrightarrow}} (w, x_1, x_2, \ldots, x_n).$$

We shall prove by induction on the number of steps in H that:

$$(\alpha_1,(\beta_{11},\pi_{11}),\ldots,(\beta_{1n},\pi_{1n})) \overset{*}{\underset{H}{\Longrightarrow}} (\alpha_2,(\beta_{21},\pi_{21}),\ldots,(\beta_{2n},\pi_{2n}))$$

if and only if $((\beta_{11},\pi_{11}),\ldots,(\beta_{1n}, \pi_{1n})) \overset{*}{\underset{G}{\Longrightarrow}} ((\beta_{21},\pi_{21}),\ldots,(\beta_{2n},\pi_{2n}))$

for some $\alpha_1, \alpha_2 \in (V_{T1} \cup V_N)^*$.

Base:

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{H}{\Longrightarrow} (\alpha_2, (\beta_{21}, P_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

if and only if $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{G}{\Longrightarrow} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$.

Let us prove it:

<u>if</u>

Suppose, $(\alpha, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{H}{\Longrightarrow} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

using the n-rule:

$$A \rightarrow (\gamma, (\delta_1, \pi_1), \ldots, (\delta_n, \pi_n)).$$

By definition of P,

$A \rightarrow ((\delta_1, \pi_1), \ldots, (\delta_n, \pi_n))$ is in P.

Therefore, $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{G}{\Longrightarrow} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$.

<u>only if</u>

Suppose, $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{G}{\Longrightarrow} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

using the n-production:

$$A \rightarrow ((\delta_1, \pi_1), \ldots, (\delta_n, \pi_n)).$$

By definition, there must exist an n-rule:

$$A \to (\gamma, (\delta_1, \pi_1), \ldots, (\delta_n, \pi_n)) \text{ in } H \text{ for some } \gamma$$

$\gamma \in (V_{T1} \cup V_N)^*$, so that,

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \underset{H}{\Rightarrow} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n})).$$

<u>induction hypothesis</u>

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{H}{\Rightarrow}} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n})),$$

if and only if $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{G}{\Rightarrow}} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$,

is true for m or less than m steps of n-derivation in H.

Now, let,

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{H}{\Rightarrow}} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

(m steps)

$$\underset{H}{\Rightarrow} (\alpha_3, (\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n})).$$

Since, $(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}) \overset{*}{\underset{H}{\Rightarrow}} (\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

if and only if $((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n})) \overset{*}{\underset{G}{\Rightarrow}} ((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$

(by induction hypothesis) and

$$(\alpha_2, (\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

$$\underset{H}{\Longrightarrow} (\alpha_3, (\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$$

if and only if

$$((\beta_{21}, \pi_{21}), \ldots, (\beta_{2n}, \pi_{2n}))$$

$$\underset{G}{\Longrightarrow} ((\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n})),$$

we have,

$$(\alpha_1, (\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$$

$$\underset{H}{\overset{*}{\Longrightarrow}} \quad (\alpha_3, (\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n}))$$

(m+1 steps)

if and only if,

$$((\beta_{11}, \pi_{11}), \ldots, (\beta_{1n}, \pi_{1n}))$$

$$\underset{G}{\overset{*}{\Longrightarrow}} ((\beta_{31}, \pi_{31}), \ldots, (\beta_{3n}, \pi_{3n})).$$

Hence the result.

CHAPTER 3

SOME PROPERTIES OF n-CONTEXT FREE LANGUAGES

In this chapter, we shall investigate some of the properties of n-context free languages. Some of the important results that will be proved are: (a) nCFL's, $n \geq 1$, define an infinite proper hierarchy of languages with CFL's at one end of the hierarchy. (b) nCFL's are recursive and their emptiness problem is recursively solvable. In section 3.1, we introduce the concept of n-direction trees. In section 3.2, we show that for some n (≥ 1), nCFL is recursive and it is decidable whether the language is empty. In section 3.3, we show that nCFG's define an infinite proper hierarchy of languages.

3.1 n-Derivation trees:

Derivation trees play an important role in the theory of context free languages. We shall extend the concept of derivation trees so as to define n-derivation trees and we shall see that n-derivation trees play a similar role in the theory of n-context free languages.

We associate with each n-derivation:

$$(A,A, \ldots, A)_n = ((\alpha_{11}, \pi_{11}), \ldots, (\alpha_{1n}, \pi_{1n})) \Rightarrow \ldots$$

$$\ldots \Rightarrow ((\alpha_{\ldots} \ldots \pi_{\ldots}), \ldots, (\alpha_{\ldots} \ldots \pi_{\ldots})).$$

in the nCFG, an ordered set of n finite rooted directed trees with labelled nodes, called an n-derivation tree.

The n-derivation tree of:

$$(A, A, \ldots, A)_n = ((\alpha_{11}, \pi_{11}), \ldots, (\alpha_{1n}, \pi_{1n})) \Longrightarrow \ldots$$
$$\ldots \Longrightarrow ((\alpha_{r1}, \pi_{r1}), \ldots, (\alpha_{rn}, \pi_{rn})),$$

is constructed as follows: The nodes of the j-th tree $1 \leq j \leq n$, in the ordered set of trees are certain tuples of the form $(i_1, \ldots, i_k)$, where $k \leq r$ and i_j is a positive integer. The directed lines of the tree are all the ordered pairs $((i_1, \ldots, i_k), (i_1, \ldots, i_k, i_{k+1}))$ of nodes. For each i, let θ_i be the n-production $A_i \rightarrow ((\alpha'_{i1}, \pi_1), \ldots, (\alpha'_{in}, \pi_n))$ such that,

$$((\beta_{i11} A_i \beta_{i12}, \pi_{i1}), \ldots, (\beta_{in1} A_i \beta_{in2}, \pi_{in}))$$
$$= ((\alpha_{i1}, \pi_{i1}), \ldots, (\alpha_{in}, \pi_{in})) \text{ and}$$

$$((\beta_{i11} \alpha'_{i1} \beta_{i12}, \pi'_{i1}), \ldots, (\beta_{in1} \alpha'_{in}, \pi'_{in}))$$
$$= ((\alpha_{i+1(1)}, \pi_{i+1(1)}), \ldots, (\beta_{i+1(n)}, \pi_{i+1(n)}))^{1}$$

The 1-tuple (1)[2] is the root of each tree in the set. The n roots of the n trees are said to n-correspond with each other form an n-node. We represent an n-node by $\{i_1, i_2, \ldots, i_n\}$. $\{1, 1, \ldots, 1\}_n$ is said to be the root of an n-derivation tree. If $\alpha_{2j} = \epsilon$, $1 \leq j \leq n$, let (1,1) be a node in the j-th tree with ϵ as the node name. If $\alpha_{2j} = X_{j21} \ldots X_{j2n(2)}$, let (1, i), $1 \leq i \leq n(2)$, be a node and X_{j2i} its node name in the j-th tree. The nodes whose node names are non-terminals - which correspond

1. The parentheses are introduced in the subscripts to avoid ambiguity.
2. In this chapter numbers in round brackets should not be confused with citation of references.

with each other in the n-form: $((\alpha_{21}, \pi_{21}), \ldots, (\alpha_{2n}, \pi_{2n}))$, are said to <u>n-correspond</u> with each other or form an <u>n-node</u>. Suppose that for all $t \leq k$, every occurrence in α_{kj}, $(1 \leq j \leq n)$, of an element of V serves as node name of some node of the j-th tree.

Now, $((\beta_{k11} A_k \beta_{k12}, \pi_{k1}), \ldots, (\beta_{kn1} A_k \beta_{kn2}, \pi_{kn}))$

$= ((\alpha_{k1}, \pi_{k1}), \ldots, (\alpha_{kn}, \pi_{kn}))$

$\Longrightarrow ((\alpha_{k+1(1)}, \pi_{k+1(1)}), \ldots, (\alpha_{k+1(n)}, \pi_{k+1(n)}))$

$= ((\beta_{k11} \alpha'_{k1} \beta_{k12}, \pi'_{k1}), \ldots, (\beta_{kn1} \alpha'_{kn} \beta_{kn2}, \pi'_{kn}))$.

Let $\{(i_{11}, \ldots, i_{1s}), \ldots, (i_{n1}, \ldots, i_{ns})\}$ be the n-node $(A_k, \ldots, A_k)_n$. If $\alpha'_{kj} = \epsilon$, for some j, $1 \leq j \leq n$, let $(i_{j1}, \ldots, i_{js}, 1)$ be a node and ϵ its node name. If $\alpha'_{kj} = X_{jk1} \cdots X_{jkn(jk)}$, let $(i_{j1}, \ldots, i_{js}, i)$, $1 \leq i \leq n\,(jk)$, be a node and X_{jki} its node name. The nodes whose node names are non-terminals which correspond with each other in the n-form: $((\alpha_{k+1(1)}, \pi_{k+1(1)}, \ldots, (\alpha_{k+1(n)}, \pi_{k+1(n)}))$, are said to <u>n-correspon</u> with each other or form an <u>n-node</u>. This procedure is repeated for all $k \leq r-1$. The resulting ordered set of finite rooted directed trees with labelled nodesx is the n-derivation tree.

The set $\{i_1, i_2, \ldots, i_n\}$ is defined as the <u>n-path</u> of the n derivation tree if the following is true:

(i) i_j is a path in the j-th tree of the n-derivation true, $i \leq j \leq n$.

(ii) If i_j, $1 \leq j \leq n$, is a sequence of nodes $n_{j1}, n_{j2}, \ldots, n_{jk}$, then for every i, $1 \leq i \leq k-1$, the nodes n_{ji}, $1 \leq j \leq n$, n-correspond with each other.

The length of each path comprising the n-path (it may be observed that these lengths will be equal)is defined as <u>length of the n-path</u>.

An <u>n-subtree</u> of an n-derivation tree is a particular n-node of the n-derivation tree together with all its descendants, the edges connecting them, and their labels.

<u>Example 3.11</u>:

Let $G = (V_N, V_T, P, S)$ be a 2CFG, where $V_N = \{S\}$, $V_T = \{a, b\}$ and P comprises of the 2-productions:

$S \rightarrow (aSb, bSa)$,

$S \rightarrow (ab, ba)$.

The 2-derivation tree of the 2-derivation:

$(S,S) \Rightarrow (aSb, bSa) \Rightarrow (aaSbb, bbSaa) \Rightarrow (aaabbb, bbbaaa)$ is given in Figure 3.11.

In the figure, node 1 is (1), node 2 is (1,1), node 3 is (1,2), node 4 is (1,3), node 5 is (1,2,1), node 6 is (1,2,2), node 7 is (1,2,3), node 8 is (1,2,2,1) and node 9 is (1,2,2,2). Similarly for the second tree of the 2-derivation tree, node 1' is (1), node 2' is (1,1), node 3' is (1,2), node 4' is (1,3), node 5' is (1,2,1), node 6' is (1,2,2), node 7' is (1,2,3), node 8' is (1,2,2,1) and node 9' is (1,2,2,2). $\{1,1'\}$, $\{3,3'\}$, $\{6,6'\}$ are all 2-nodes.

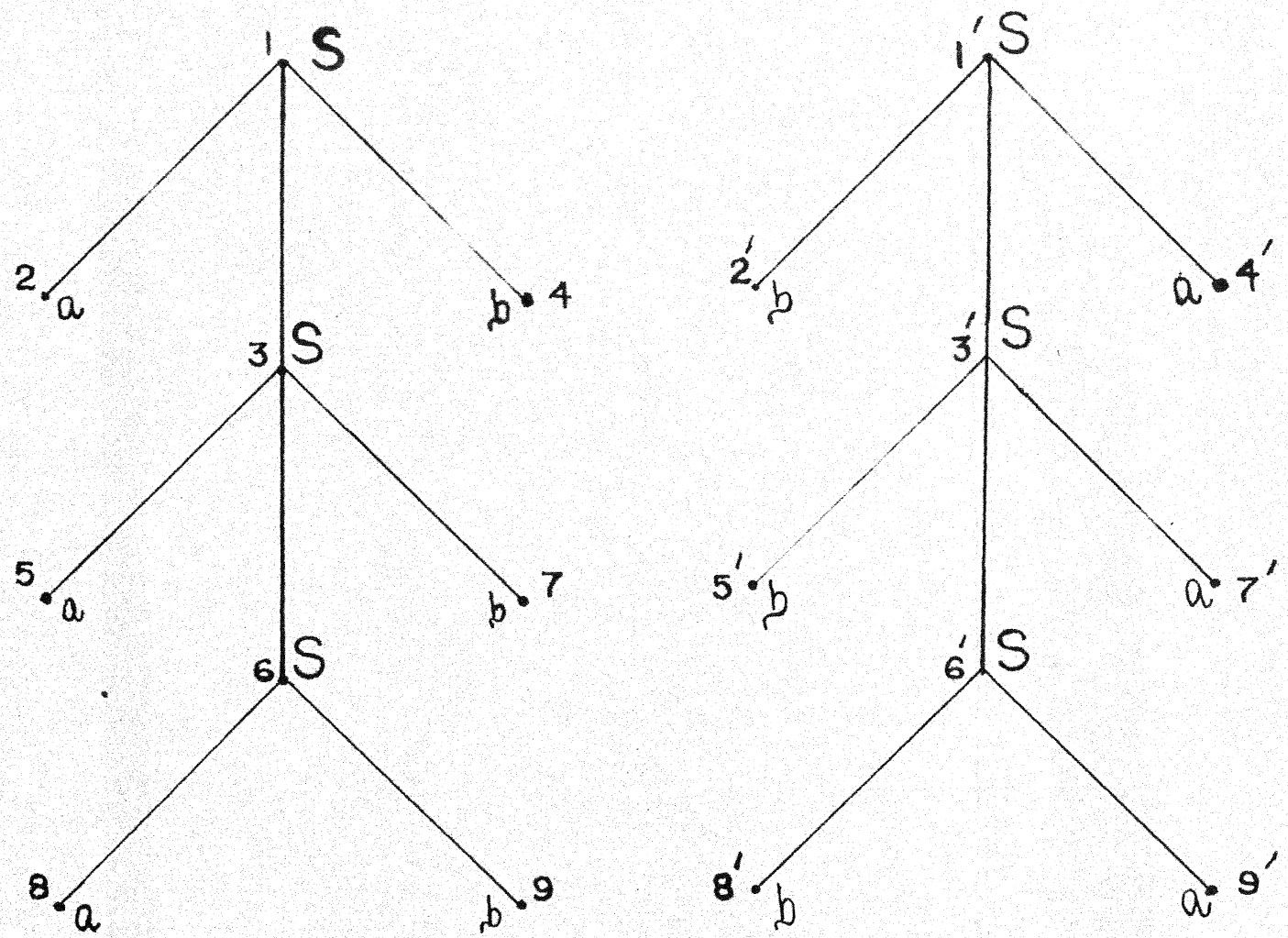


FIGURE 3·11

3.2 Recursiveness and emptiness problems:

As a prelude to proving the required theorems, we shall give some lemmas:

It will be observed that as we have defined nCFG, the empty sentence ϵ cannot be in any nCFL. We shall extend the definition of nCFG to allow n-productions of the form: $S \rightarrow (\epsilon, \epsilon, \ldots \epsilon)_n$, where S is the start symbol, provided that S does not appear in the output expansion of any n-production. In that case, it is clear that the n-production $S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)_n$ can only be used as the first step in an n-derivation.

Lemma 3.21:

If $G = (V_N, V_T, P, S)$ is an nCFG, then there is another nCFG G', for which the start symbol of G' does not appear in the output expansions of any n-production of G.

Proof:

Let S_1 be a symbol not in V. Let $G' = (V_N', V_T, P', S_1)$ where $V_N' = V_N \cup S_1$ and P' consists of all n-productions of P plus all n-productions of the form $S_1 \rightarrow ((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$, where $S \rightarrow ((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ is an n-production of P.

It is easy to see that $L(G) = L(G')$.

Lemma 3.22:

If L is an nCFL, then $L \cup \{\epsilon\}$ and $L - \{\epsilon\}$ are also nCFL.

Proof:

Given an nCFG, we can find an equivalent nCFG G whose start symbol does not appear in the output expansion of any

n-production. Let $G = (V_N, V_T, P, S)$. Define $G' = (V_N, V_T, P', S)$, where P' is P plus the n-production:

$$S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)_n.$$

Note that S does not appear in the output expansion of any n-production of P'. Thus $S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)_n$ cannot be used in a manner other than as the first and only n-production in an n-derivation. Any n-derivation in G' not involving $S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)_n$ is an n-derivation in G. Therefore, $L(G') = L(G) \cup \{\epsilon\}$.

Again, if the nCFG $G = (V_N, V_T, P, S)$ generates the empty sentence ϵ, the start symbol S must not appear in the output expansion of any n-production and the n-production $S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)$ must be in P. Define $G' = (V_N, V_T, P', S)$, where P' is P minus the n-production: $S \rightarrow (\epsilon, \epsilon, \ldots, \epsilon)_n$. Clearly $L(G') = L(G) - \{\epsilon\}$.

Lemma 3.23:

Let $G = (V_N, V_T, P, S)$ be an nCFG. Let $((\beta_1, \pi_1), \ldots (\beta_n, \pi_n))$ be an n-form such that $|\beta_1, \ldots \beta_n| > np$. Then $(\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ cannot derive $(w_1, \ldots, w_n)$ such that $|w_1 w_2 \ldots w_n| \leq p$.

Proof:

Let $|\beta_1 \ldots \beta_n| = t$, $(> np)$.

There can be two possibilities:

(i) The number of non-terminals in β_j's, $1 \leq j \leq n$, $\leq p$.

(ii) The number of non-terminals in β_j's $> p$.

If (i) is the case, let the number of non-terminals in β_j's be k, ($\leq$ p), so that the number of terminals in $\beta_1\beta_2 \ldots \beta_n$ is t-nk > 0.

Now, the only way in which,

$((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n)) \overset{*}{\underset{G}{\Longrightarrow}} ((\gamma_1, \pi_1'), \ldots, (\gamma_n, \pi_n'))$

s.t. $|\gamma_1\gamma_2 \ldots \gamma_n| < |\beta_1\beta_2 \ldots \beta_n|$, is by using n-productions of the type: $A \rightarrow ((\delta_1, \pi_1''), \ldots, (\delta_n, \pi_n''))$, where one or more than one δ_j, $1 \leq j \leq n = \epsilon$. Let r non-terminals, ($\leq$ k), be reduced to terminals using such n-productions. Then, $|\gamma_1\gamma_2 \ldots \gamma_n| \geq$ t-np+p because there has to be atleast one symbol other than ϵ in the output expansion of an n-production.

Since, t-np > 0, $|\gamma_1\gamma_2 \ldots \gamma_n| > p$ and hence the result.

If (ii) is the case, the result is immediate. Therefore, $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))|_{|\beta_1 \ldots \beta_n| > np}$ cannot derive $(w_1, \ldots, w_n)|_{|w_1 \ldots w_n| \leq p}$.

Theorem 3.21:

If $G = (V_N, V_T, P, S)$ is an nCFG, $n \geq 1$, then G is recursive.

Proof:

We give an algorithm for finding whether a sentence $w|_{|w|=p} \in L(G)$.

Define the set T_m as the set of n-forms[1] $((\beta_1,\pi_1),\ldots,(\beta_n,\pi_n))$ where, $|\beta_1\beta_2 \ldots \beta_n| \leq np$ and

$$(S, S, \ldots, S)_n \overset{*}{\underset{G}{\Longrightarrow}} ((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$$

by an n-derivation of atmost m steps. Clearly, $T_o = \{(S,S,\ldots,S)_n\}$.

It is easy to see that we can calculate T_m from T_{m-1} by seeing what n-forms $((\gamma_1, \pi_1), \ldots,(\gamma_n, \pi_n))|_{|\gamma_1\gamma_2\ldots\gamma_n| \leq np}$ can be derived from n-forms in T_{m-1} by a single application of an n-production.

Also, if $(S, S, \ldots, S)_n \overset{*}{\underset{G}{\Longrightarrow}} (w_1, w_2, \ldots, w_n)$ and $|w_1 \ldots w_n| \leq p$, then $(w_1, \ldots, w_n)$ will be in T_m for some m. This is because $(w_1, w_2, \ldots, w_n)|_{|w_1w_2\ldots w_n|\leq p}$ cannot be derived from an n-sentential form $((\beta_1,\pi_1),\ldots,(\beta_n,\pi_n))|_{|\beta_1\beta_2\ldots\beta_n| > np}$. Therefore, any n-sentential form which can derive $(w_1,w_2,\ldots,w_n)$ will be in T_m for some m and hence $(w_1, w_2, \ldots, w_n)$ will be in T_m for some m.

It should also be evident that $T_m \supseteq T_{m-1}$ for all $m \geq 1$. Since T_m depends only on T_{m-1}, if $T_m = T_{m-1}$, then $T_m=T_{m+1} = T_{m+2} = \ldots$ Our algorithm will be to calculate $T_1, T_2, \ldots$ until for some m, $T_m = T_{m-1}$. If there is no $(w_1, w_2, \ldots, w_n)$ in T_m such that $w = w_1w_2 \ldots w_n$, then w is not in L(G) because for $j > m$, $T_j = T_m$.

The only thing that remains to be shown is: for some m $T_m = T_{m-1}$. Since $T_i \supseteq T_{i-1}$, if $T_i \neq T_{i-1}$, then the number of n-forms in T_i is atleast one greater than the number in T_{i-1}.

---

1. The π's, in these n-forms are there only to indicate the correspondence between the non-terminals. These will be dropped if there is no ambiguity in this correspondence.

Let the number of symbols in V be y. Then the number of n-forms $((\beta_1, \pi_1), \ldots, (\beta_n, \pi_n))$ $|\beta_1\beta_2 \ldots \beta_n| \leq np$ is less than $ny + (ny)^2 + \ldots + (ny)^{np}$ and hence is less than $(ny+1)^{np+1}$.

Hence the result.

Now we shall take up the emptiness problem of nCFL.

Theorem 3.22:

There is an algorithm for determining if the language generated by a given 2CFG is empty.

Proof:

Let $G = (V_N, V_T, P, S)$ be a 2CFG. Suppose that $(S, S) \overset{*}{\underset{G}{\Rightarrow}} (w_1, w_2)$ for some terminal string $w = w_1 w_2$. Consider the 2-derivation tree of w in the grammar G. Suppose that there is a 2-path in the 2-derivation trees, such that in each path of the 2-path, there are two nodes n_1 and n_2, having the same label A, with n_1 higher on the path than n_2.

The 2-derivation tree for the 2-drivation of w = aabaacbaabcdd in the 2-CFG:

$$G = (\{S,A,B\}, \{a,b\}, \{S \to (Aa, cbA),\ A \to (aB, Bd),\ B \to (Aa, aA),\ A \to (aB, aBd,\ B \to (b, bc)\}, S)$$

is given in Figure 3.21.

The 2-subtree with root at $\{n_1, n_1\}$ represents the generation of a word $w_{11}\ w_{21}$, such that

$$(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w_{11}, w_{21}).$$

The 2-subtree with root at $\{n_2, n_2\}$ likewise represents the

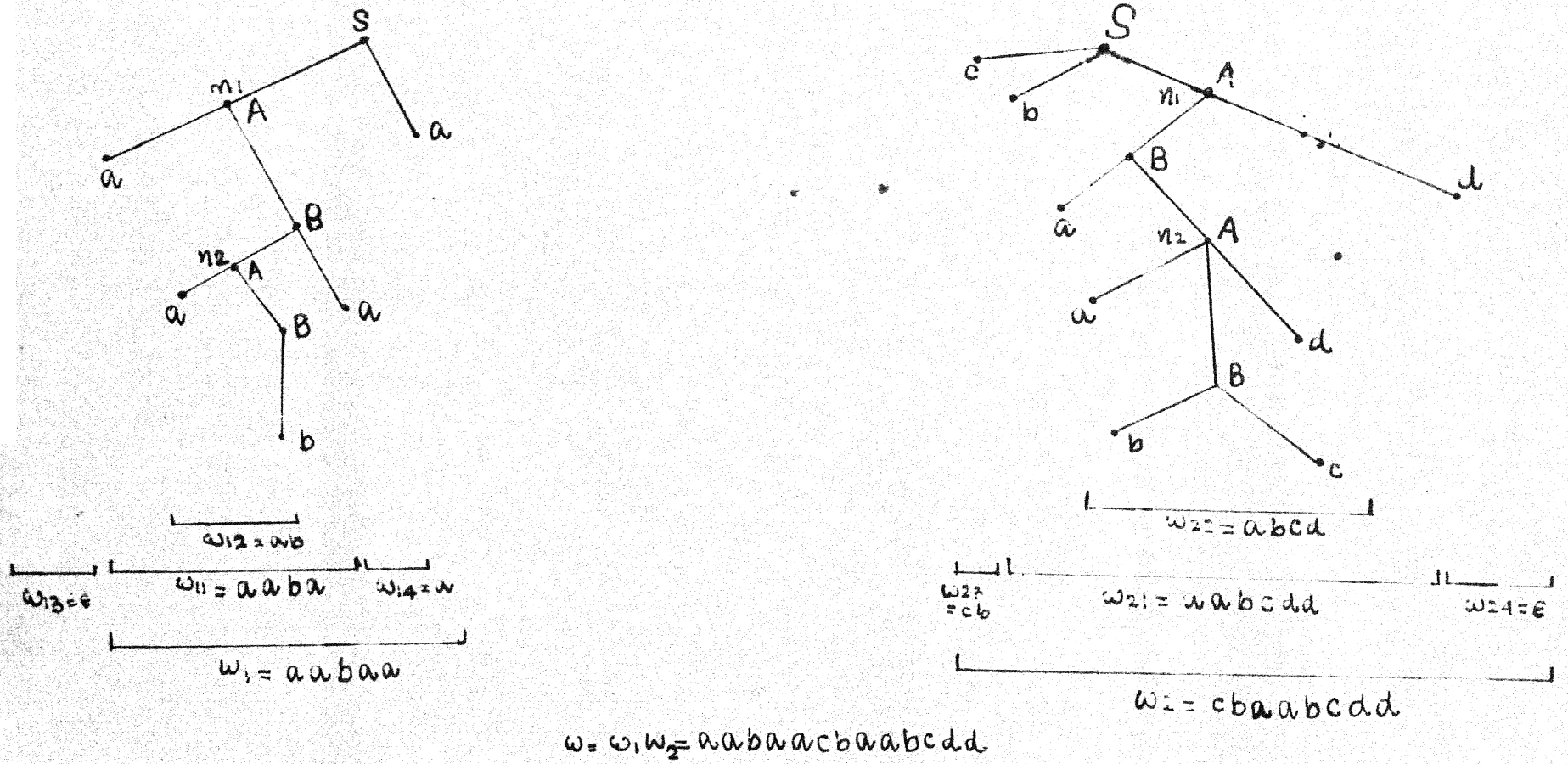
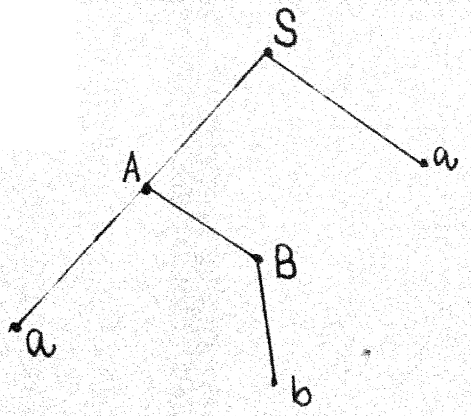

S
n_1
A
a
a
B
n_2
A
a
B
a
b
$\omega_{12} = ab$
$\omega_{13} = \epsilon$
$\omega_{11} = aaba$
$\omega_{14} = a$
$\omega_1 = aababa$
S
c
b
A
n_1
d
B
a
n_2
A
a
d
B
b
c
$\omega_{22} = abcd$
$\omega_{23} = cb$
$\omega_{21} = aabcdd$
$\omega_{24} = \epsilon$
$\omega_2 = cbaabcdd$
$\omega = \omega_1\omega_2 = aababacbaabcdd$

FIGURE 3.21



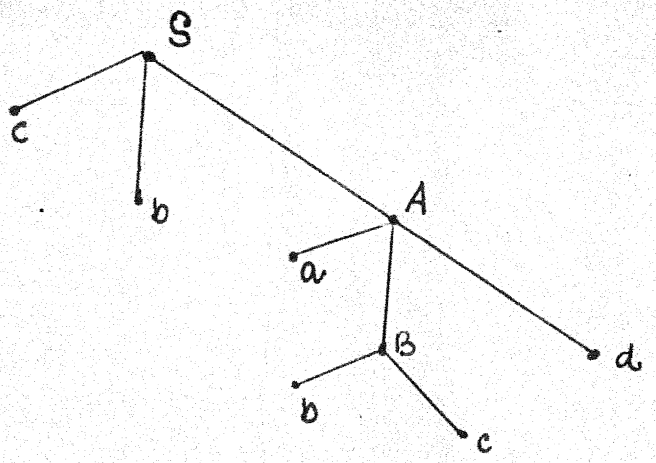


FIGURE 3·22

generation of a word $w_{12}\ w_{22}$, such that $(A, A) \overset{*}{\underset{G}{\Longrightarrow}} (w_{12}, w_{22})$. (Note w_{12} must be a subword of w_{11} and w_{22} must be a subword of w_{12}). Now the word w can be written in the form $w_{13}\ w_{11}\ w_{14} w_{23} w_{21} w_{24}$ where some or all among w_{13}, w_{14}, w_{23}, w_{24} may be ϵ. If we replace the 2-subtree at $\{n_1, n_1\}$ by that at $\{n_2, n_2\}$, we have a new word, $w_{13}\ w_{12}\ w_{14}\ w_{23}\ w_{22}\ w_{24}$ (possibly the same word), s. t.

$$(S, S) \overset{*}{\underset{G}{\Longrightarrow}} (w_{13}\ w_{12}\ w_{14}, w_{23}\ w_{22}\ w_{24}).$$

In the above figure,

$$w_{13} = \epsilon,\ w_{14} = a,\ w_{23} = cb,\ w_{24} = \epsilon$$

2-derivation tree for the 2-derivation,

$(S, S) \overset{*}{\underset{G}{\Longrightarrow}} (w_{13}\ w_{12}\ w_{14}, w_{23}\ w_{22}\ w_{24})$ is shown in Figure 3.22.

However, we have eliminated atleast one node, n_1 from each tree in the 2-derivation tree of Figure 3.21. If the new 2-derivation tree has a 2-path with two identically labelled nodes in each path comprising the 2-path, the process may be repeated with $w_{13}\ w_{12}\ w_{14}\ w_{23}\ w_{22}\ w_{24}$ instead of w. In fact, the process may be repeated, until there are no 2-paths in the 2-derivation tree with two nodes in each path named identically. Since each iteration eliminates one or more nodes from each tree of the 2-derivation tree, the process must eventually terminate.

Now consider the 2-derivation tree which is ultimately produced. If there are m non-terminals in the grammar G, then there can be no 2-path of length greater than m, lest some non-terminal should be repeated in the two paths comprising the 2-path.

We conclude that if G generates any word at all, then there is a 2-derivation of a word whose 2-derivation tree contains no 2-path of length greater than m. Thus the following algorithm with determine if G is empty:

Form a collection of 2-derivation trees corresponding to 2-derivations in G as follows: Start the collection with the 2-derivation tree containing the single 2-node $\{S, S\}$. Repeatedly add to the collection any 2-derivation tree that can be obtained from a 2-derivation tree already in the collection by application of a single 2-production and that:

(i) is not already in the collection.

(ii) does not have any 2-path of length greater.

The process will clearly terminate. Now, L(G) is non empty if and only if atleast one of the 2-derivation trees in the collection corresponds to the 2-derivation of a terminal string.

Cor. 3.221:

There is an algorithm for determining if the language generated by a given nCFG, $n \geq 1$, is empty.

3.3 uvwxy theorem for n-context free languages:

First, let us prove some auxiliary lemmas:

Lemma 3.31: (Normal form for nCFG's)

Any nCFL can be generated by an nCFG in which all n-productions are of the form:

$A \rightarrow ((B_1, \pi_1), \ldots, (B_n, \pi_n))$ or $A \rightarrow (b_1, b_2, \ldots, b_n)$, where A, B_i, $1 \leq i \leq n$, are non-terminals and b_i, $1 \leq i \leq n$, $\in V_T^*$ such that $b_1 b_2 \ldots b_n \neq \epsilon$.

Proof:

Let $G = (V_N, V_T, P, S)$ be any nCFG. We shall construct another nCFG $G' = (V_N', V_T, P', S)$ as follows:

To start with V_N' contains all the non-terminals in V_N and the set P' contains no n-production. For each n-production:

$$A \rightarrow ((w_{11}A_1 \cdots w_{1m}A_m w_{1(m+1)}, \pi_1), (w_{21}B_{21} \cdots w_{2m}B_{2m}w_{2(m+1)}, \pi_2$$

$$\cdots, (w_{n1}B_{n1} \cdots w_{nm}B_{nm}w_{n(m+1)}, \pi_n)),$$

in P, add to V_N' the non-terminals $C_1, C_2, \ldots, C_{m+1}$ and $D_1, D_2, \ldots, D_{m+1} \notin V_N'$ if $w_{1(m+1)}\ w_{2(m+1)} \cdots w_{n(m+1)} \neq \epsilon$, otherwise add to V_N' the non-terminals $C_1, C_2, \ldots, C_{m+1}$ and $D_1, D_2, \ldots, D_m \notin V_N'$.

Add to P the following n-Productions:

1. (i) $A \rightarrow (C_{m+1}\ D_{m+1}, \ldots, C_{m+1}\ D_{m+1})_n$

(ii) $D_{m+1} \rightarrow (w_{1(m+1)}, \ldots, w_{n(m+1)})$,

if $w_{1(m+1)} \cdots w_{n(m+1)} \neq \epsilon$.

otherwise add:

(i) $A \rightarrow (C_{m+1}, \ldots, C_{m+1})_n$

2. $C_{m+1} \rightarrow ((C_1 \cdots C_m, \pi_1), (C_{\hat{\pi}_2(\pi_1(1))} \cdots C_{\hat{\pi}_2(\pi_1(m))}, \pi_2), \ldots$

$\ldots, (C_{\hat{\pi}_n(\pi_1(1))} \cdots C_{\hat{\pi}_n(\pi_1(m))}, \pi_n))$,

3. $C_j \rightarrow (D_j A_j, \ldots, D_j A_j)_n$, for $1 \leq j \leq m$.

4. $D_j \rightarrow (w_{1j}, w_{2\pi_2(\hat{\pi}_1(j))}, \ldots, w_{n\pi_n(\hat{\pi}_1(j))})$, $1 \leq j \leq m$

From the construction it is evident that $L(G) = L(G_1)$. We shall not give an exhaustive proof for it.

Now we shall state and prove the uvwxy theorem for nCFL's.

Theorem 3.31:

Let L be any 2CFL. There exist constants p and q depending only on L, such that if there is a word z in L, with $|z| > 2p$, then z may be written as,

$$z = u_1 v_1 w_1 x_1 u_2 v_2 w_2 x_2 y_2,$$

where $|v_1 w_1 x_1|, |v_2 w_2 x_2| \leq q$

and $v_1 x_1 v_2 x_2 \neq \epsilon$, such that for each integer $i \geq 0$,

$u_1 v_1^i w_1 x_1^i u_2 v_2^i w_2 x_2^i y_2$ is in L.

Proof:

Let $G = (V_N, V_T, P, S)$ be any normal-form 2CFG for L. If G has k variables and is of order l, then let $p = l^{k-1}$ and let $q = l^k$. It is easy to see that, for a normal form grammar, if an 2-derivation tree has no 2-path longer than j, then the terminal string derived is no longer than $2(l^{j-1})$.

Hence if z is in L and $|z| > 2p$, then the 2-derivation tree for any 2-derivation of z by the grammar G contains a 2-path of length greater than k. We consider a 2-path P, of longest length, and observe that there must be two 2-nodes $\{n_1, n_1\}$ and

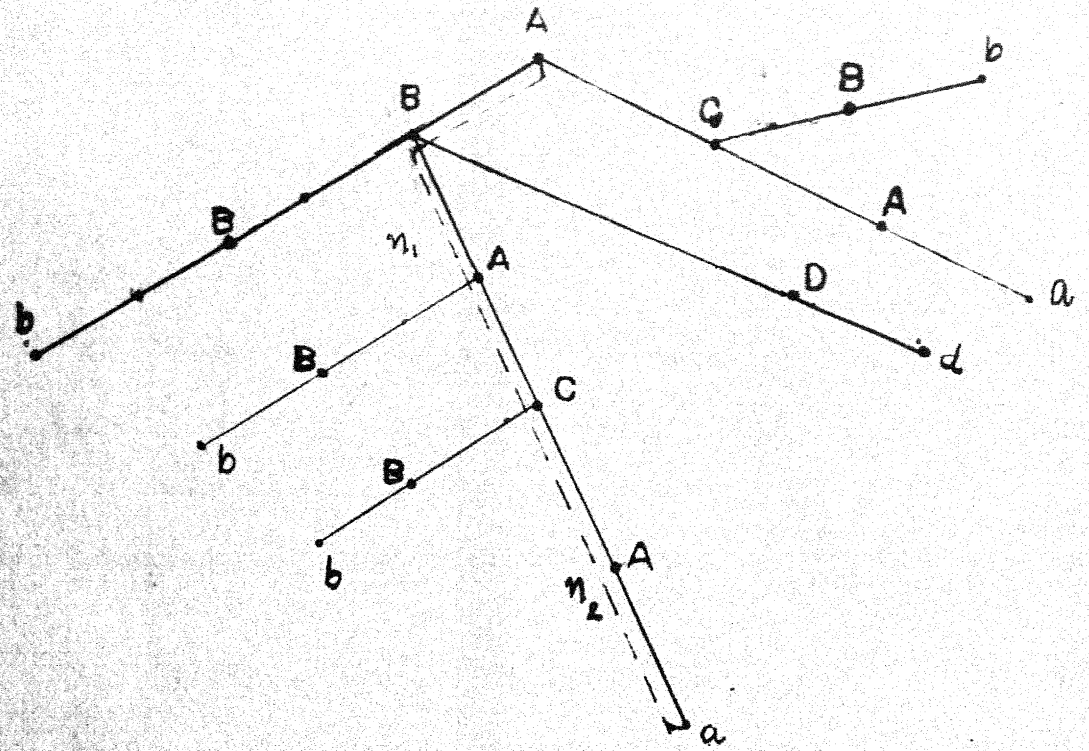
$\{n_2, n_2\}$, satisfying the following conditions:

In each tree of the 2-derivation tree:

1. The nodes n_1 and n_2 both have the same label, say A.
2. Node n_1 is closer to the root than node n_2.
3. In the path which forms part of the 2-path P, length from n_1 to the leaf is atmost $k + 1$.

To see that n_1 and n_2 can always be found, for each tree comprising the 2-derivation tree, just proceed up the path corresponding to the 2-path P, from the leaf, keeping track of the labels encountered. Of the first $k + 2$ nodes, only the leaf has a terminal label. The remaining k+1 cannot have distinct variable labels.

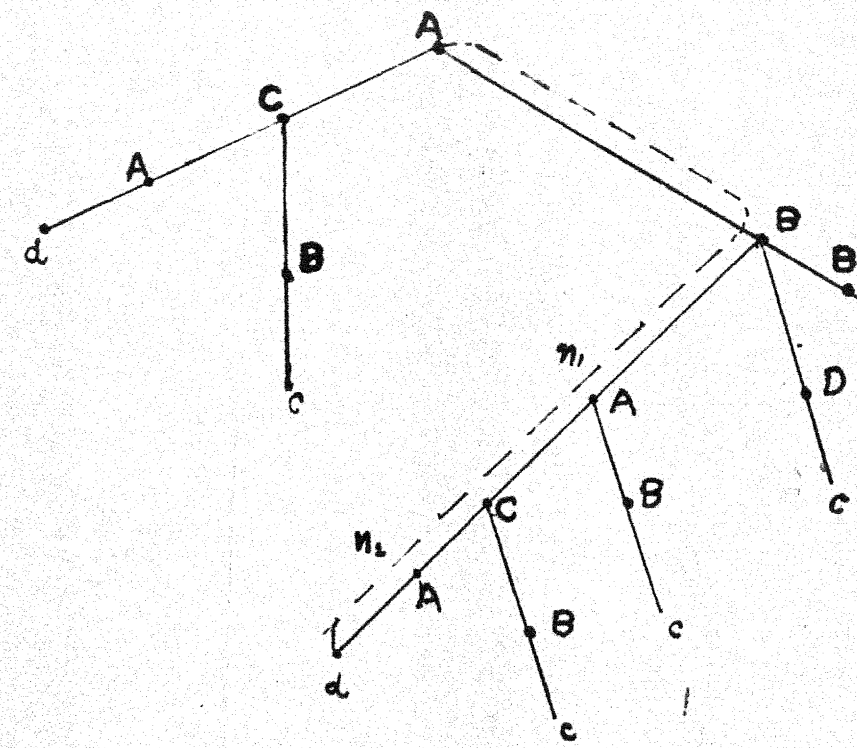
Let us look at the 2-subtree with root $\{n_1, n_1\}$ in Figure 3.31. Each subtree T_{11}, T_{21} comprising the 2-subtree represents the derivation of a subword of length atmost 1^k (and hence of length $\leq q$). This is true because there can be no path in T_{11} or T_{21} of length greater than $k + 1$, since P comprises of longest paths in the two trees of the 2-derivation tree. Let z_{11} be the result of subtree T_{11}. Similarly considering the 2-subtree with root $\{n_2, n_2\}$, let T_{12} and T_{22} be the two subtrees comprising it. Let z_{12} be the result of T_{12}. Then we can write z_{11} as $z_{13}\ z_{12}\ z_{14}$. Similarly, we can write z_{21} as $z_{23}\ z_{22}\ z_{24}$. Further $z_{13}\ z_{14}\ z_{23}\ z_{24}$ cannot be ϵ (because these cannot be any 2-production of the form $A \rightarrow (\epsilon, \epsilon)$ and we can remove 2-productions of the form $A \rightarrow (B, B)$ from the grammar).



$\gamma_{13} = bb$ $\quad \gamma_{12} = a$ $\quad \gamma_{14} = \epsilon$

$\gamma_{11} = bba$

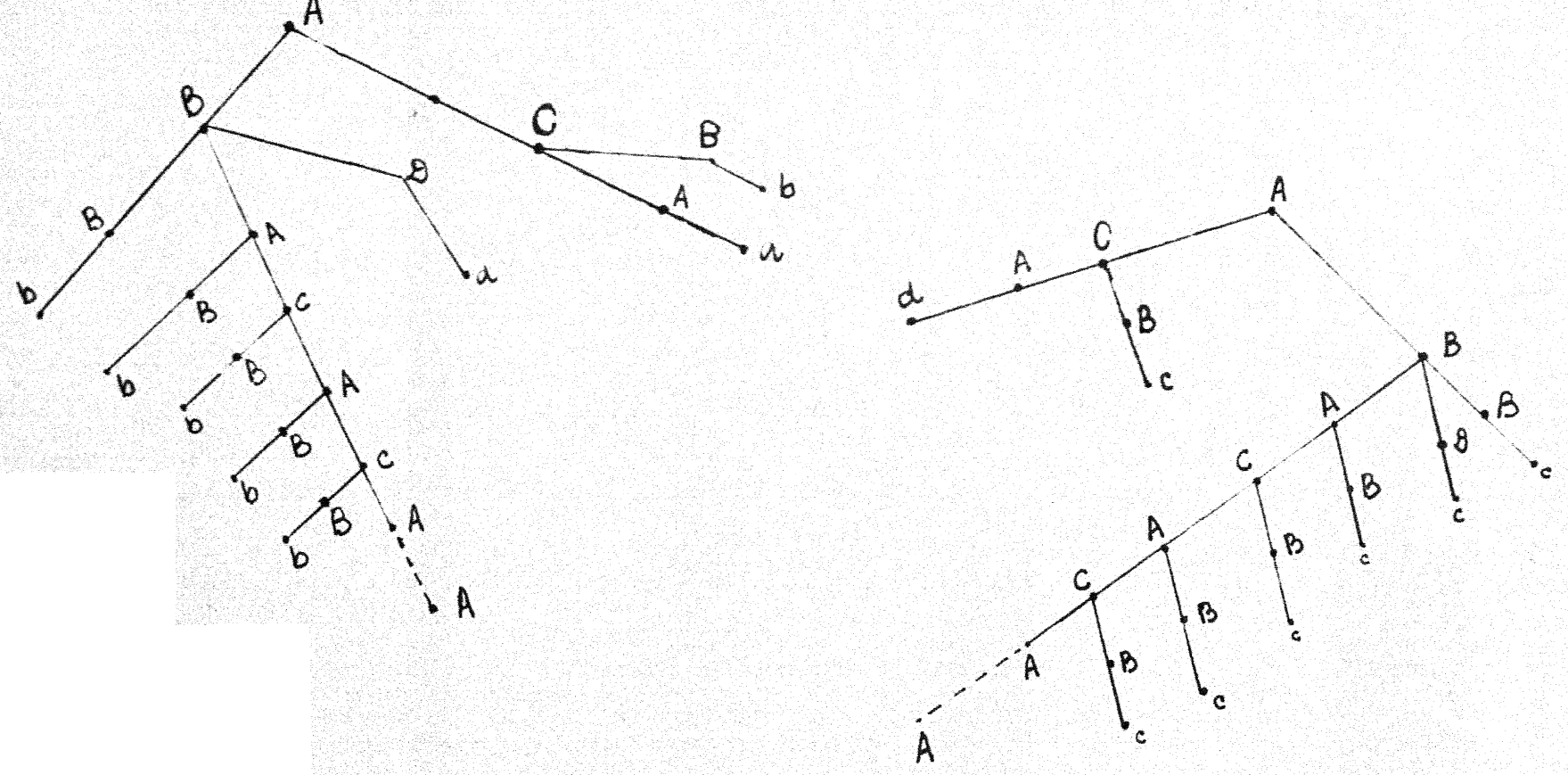
$\gamma_1 = bbb\,a\,d\,a\,b$



$\gamma_{23} = \epsilon$ $\quad \gamma_{22} = d$ $\quad \gamma_{24} = cc$

$\gamma_{21} = dcc$

$\gamma_2 = dcdcccc$



2-DERIVATION OF $U_1V_1W_1X_1U_2V_2W_2X_2Y_2$

FIGURE 3·32

We know that $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (z_{13}Az_{14}, z_{23}Az_{24})$

$\overset{*}{\underset{G}{\Rightarrow}} (z_{13}z_{12}z_{14}, z_{23}z_{22}z_{24})$,

where $z_{13}z_{12}z_{14}, z_{23}z_{22}z_{24} \leq q$.

But then it follows that,

$(A, A) \overset{*}{\underset{G}{\Rightarrow}} (z_{13}{}^{i}z_{12}z_{14}{}^{i}, z_{23}{}^{i}z_{22}z_{24}{}^{i})$, for each $i \geq 0$.

The string $z = z_1z_2$ can clearly be written as

$u_1z_{13}z_{12}z_{14}u_2z_{23}z_{22}z_{24}y_2$, for some u_1, u_2, y_2. We let $z_{13} = v_1, z_{12} = w_1, z_{14} = x_1, z_{23} = v_2, z_{22} = w_2, z_{24} = x_2$ to complete the proof.

Cor. 3.311:

Let L be any nCFL. There exist constants p and q depending only on L, such that if there is a word z in L, with $|z| > np$, then z may be written as $z = u_1v_1w_1x_1 \ldots u_nv_nw_nx_ny_n$, where $|v_iw_ix_i|$, $1 \leq i \leq n$, $\leq q$ and $v_1x_1 \ldots v_nx_n \neq \epsilon$, such that for each integer $i \geq 0$.

$u_1v_1{}^{i}w_1x_1{}^{i} \ldots u_nv_n{}^{i}w_nx_n{}^{i}y_n$ is in L.

Using the above theorem, we shall now prove an important result which establishes the infinite proper hierarchy of nCFL.

Theorem 3.32:

$L = \{a_1{}^{k}a_2{}^{k} \ldots a_{2n+1}^{k} \mid_{k \geq 1}\}$, is not nCFL.

Proof:

Let $L = \{a_1{}^{k}a_2{}^{k} \ldots a_{2n+1}^{k} \mid k \geq 1\}$ be an nCFL. Then by Cor. 3.311, there exist constants p and q such that

$z = a_1^p a_2^p \ldots a_{2n+1}^p$ can be written as,

$z = u_1v_1w_1x_1 \ldots u_nv_nw_nx_ny_n$, $|v_iw_ix_i|$, $1 \leq i \leq n$, $\leq q$ and $v_1x_1 \ldots v_nx_n \neq \epsilon$ such that for each integer $i \geq 0$,

$u_1v_1^iw_1x_1^i \ldots u_nv_n^iw_nx_n^iy_n$ is in L.

It follows that all of $v_1, x_1, \ldots, v_n, x_n$ cannot be empty. Also, since all symbols $a_1, a_2, \ldots, a_{2n+1}$ are distinct, any of $v_1, x_1, \ldots, v_n, x_n$ cannot contain more than one symbol. Suppose out of these r ($\leq 2n$) v_i's and x_i's contain one distinct symbol each and the rest are empty. Let the other variables $u_1, w_1, u_2, w_2, \ldots, u_n, w_n, y_n$ contain symbols of total length s.

According to the theorem,

$u_1v_1^{2s}w_1x_1^{2s} \ldots u_nv_n^{2s}w_nx_n^{2s}y_n$ is in L.

We note that

$|u_1v_1^{2s}w_1x_1^{2s} \ldots u_nv_n^{2s}w_nx_n^{2s}y_n| \leq 2ns + s.$

But, since not all $v_1, x_1, \ldots, v_n, x_n$ can be empty and any of these can atmost contains one distinct symbol, the length of sentence represented by $u_1v_1^{2s}w_1x_1^{2s} \ldots u_nv_n^{2s}w_nx_n^{2s}y_n$ cannot be less than $2s(2n+1) = 4ns + 2s$ - a contradiction.

Hence the result.

Cor. 3.321:

nCFG properly contains (n-1) CFG.

This is because $\{a_1^k a_2^k \ldots a_{2n+1}^k \mid k \geq 1\}$ is obviously an (n+1)CFL while it is not nCFL and any nCFL is trivially an (n+1)CFL.

CHAPTER 4

CONCLUSIONS

In this chapter we shall conclude our work with making certain conjectures and suggesting the lines along which further work can be done. In section 4.1, we make certain conjectures. In section 4.2, we give some concluding remarks and suggestions for future work.

4.1 Some conjectures:

We shall now give some conjectures which we feel can be proved:

(i) If a translation T is an nSDT of order k, then $T = \tau(M)$[1] for some n(k)APA[1] M.

(ii) $T = \tau(M)$[1] for an n(k)APA[1] $M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \rho, q_0, Z_0)$, then T = T(H) for an nSDTG $H = (V_N, V_{T1}, V_{T2}, R, S)$ of order k.

(iii) If T is an nSDT of order 2k, then T is defined by an n(k)BPA[1] M.

(iv) A translation T is definable by a 1-register n-type B pushdow assembler[1] (n(1)BPA) if and only if T is an nSSDT.

The above conjectures are concerning the relationship between nSDT's and n-pushdown assemblers[1]. Since we have nonstructed n-pushdown assemblers as a natural extension of pushdown assemblers,

1. For details of n-pushdown assemblers, see Appendix B.

we feel that the corresponding proofs for pushdown assemblers can be extended for proving these conjectures.

Now we give some conjectures concerning n-context free languages:

(v) Ibarra (6) has defined a class of languages called 'Simple matrix languages'. We feel that simple matrix grammar of order n is equivalent to n-simple context free grammar.

(vi) Context sensitive languages properly contain $\bigcup_{n=1}^{\infty}$ nCFL. In particular, $L = \left\{ a^n b^{n^2} \mid_{n \geq 1} \right\}$ cannot be generated by any nCFG.

(vii) nCFG properly contains nCFG.

There are many properties of n-context free languages like (a) The finiteness or infiniteness problem is decidable. (b) These languages are closed under union and substitution mapping. But we have not reported these results in this work because there is no point giving some stray properties about these languages unless some fundamental questions are answered which we will discuss in the next section.

4.2 Concluding remarks and suggestions for future work:

In presenting a class of models for translations which we have called n-syntax directed translation grammars, we were motivated by a desire to have models which are sufficiently powerful as well as simple. We feel that we have succeeded to some extent in that respect.

After presenting these models, our aim, in this work, has been to investigate these models from automata theoretic point of view. Out of the many features of these models we chose to do

some investigation on the range languages of these translations which we have called n-context free languages.

In the process of investigating these range languages of nSDT's we have found that these languages are sufficiently interesting to deserve a thorough investigation in their own right. The interesting features of these languages are that they are in fact a proper hierarchy of languages in between context free languages and context sensitive languages (a conjucture). Such languages are of current interest (See 1,6,7), because for many purposes we are finding that while context free grammars are quite simple to use, they are not sufficiently powerful and while context sensitive grammars are quite powerful, they are difficult to use.

What we have not done in this work and what is sufficiently interesting to work on is the following:

(a) How do we characterise the class of n-context free languages? This is an important question because a new type of grammar for a language will not serve much purpose unless we get a detailed idea of what sort of languages will be generated by this type of grammar. Specifically, given a language, there must be some way (in fact, there should be more than one way) of knowing whether this language can be generated by the given type of grammar. Such questions for types of grammars already in vogue have been answered by giving a machine characterisation of the languages generated by these grammars. They have also been answered by investigating thoroughly the closure properties of languages generated by these grammars.

(b) Another thing which needs to be looked into is the relationship of nCFG's with other types of grammars or classes of grammars that have been proposed recently. In particular, work of Takuni Kasai (7) and Ibarra (6) will be of interest, in this context. As stated earlier, we feel that the classes of grammars proposed by Ibarra is equivalent to nSCFG's.

(c) Coming back to translation models, where from we started our work, some investigation needs to be done about the practical usefulnes of nSDT's. For a translation model to be practically useful, it needs to satisfy a complex balance between power (of the model) and simplicity. Only after making a working system based on this model, will one get a correct idea about its usefulness.

REFERENCES

1. Aho, A.V.: 'Indexed grammars - an extension of context-free grammars', J.ACM 15 (October 1968), pp. 647-671.

2. Aho, A.V. and Ullman, J.D.: 'Automaton analogs of syntax directed translation schemata', IEEE Conference Record of Ninth Annual Symposium on Switching and Automata Theory, October 1968, pp. 143-159.

3. Aho, A.V. and Ullman, J.D.: 'Translations on a context-free grammar', ACM Symposium on Theory of Computing (1969), pp. 93-112.

4. Feldman, J. and Gries, D.: 'Translation writing systems', Comm. ACM 11 (February 1968), pp. 77-113.

5. Hopcroft, J.E. and Ullman, J.D.: 'Formal languages and their relation to automata', Addison-Wesley Publishing Co. (1969).

6. Ibarra, O.H.: 'Simple matrix languages', Information and control 17, pp. 359-394 (1970).

7. Kasai, T.: 'An hierarchy between context-free and context-sensitive languages', Journal of Computer and System Sciences, 4, pp. 492-508 (1970).

8. Lewis, P.M. II and Stearns, R.E.: ' Syntax directed transduction', J.ACM 15 (July 1968), pp. 464-488.

9. Petrone, L.: 'Syntax directed mappings of context free languages', IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory, October 1968, pp. 160-175.

10. Sahasrabuddhe, H.V.: '2-grammars', Proceedings of the 6th Annual Conference of Computer Society of India, January 1971.

11. Stearns, R.E. and Lewis II, P.M.: 'Property languages and table machines', IEEE Conference Record of 9th Annual Symposium on Switching and Automata Theory, October 1968, pp. 106-119.

APPENDIX A

Here, we shall give a 4-syntax directed translation grammar (augmented by two counters m and n) which will describe the translation discussed in Introduction.

$$H = (V_N, V_{T1}, V_{T2}, R, S)$$

where $V_N = \{S, S_1, S_2, L, L_1, L_2, E, M\}$,

$V_{T1} = \{A, B, C, D, E, \not{\emptyset}, \text{READ}, -, *, \text{PUNCH}\}$,

$V_{T2} = \{A, B, C, D, E, \not{\emptyset}, \not{\#}, \text{DORG}, \text{DEFINE}, \text{LOAD}, \text{SUB}, \text{MPY}, \text{ST}\not{\emptyset}, \text{END}\}$,

and R comprises of the following 4-rules[1]:

1. $S \rightarrow (S_1 \not{\#} L_2 = E \not{\#} S_2 \not{\#} \text{END};\ (\text{DORG}\not{\emptyset}400 \not{\#} \text{DEFINE}\not{\emptyset} L_2 \not{\#} S_1 E S_2);$
   $(S_1 L_1 E S_2);(\ m=1\ S_1 E L_2 S_2);(\ n=1\ S_1 E\ \text{ST}\not{\emptyset}\not{\emptyset} L_2 \not{\#} S_2 \text{END} \not{\#}))$
2. $S_1 \rightarrow (\text{READ}\not{\emptyset} L; L; L; L; L)$
3. $L \rightarrow (L_3, L; (\text{DEFINE}\not{\emptyset} L_3 \not{\#} L);\ (\text{READ}\not{\emptyset} L_3 \not{\#} L); (L_3 L); (L_3 L))$
4. $L_3 \rightarrow (A; A; A; \varphi; \varphi)$
5. $L_3 \rightarrow (B; B; B; \varphi; \varphi)$
6. $L_3 \rightarrow (C; C; C; \varphi; \varphi)$
7. $L_3 \rightarrow (D; D; D; \varphi; \varphi)$
8. $L_3 \rightarrow (E; E; E; \varphi; \varphi)$
9. $L \rightarrow (A; (\text{DEFINE}\not{\emptyset} A \not{\#}); (\text{READ}\not{\emptyset} A \not{\#});\ (\varphi),\ (\varphi))$
10. $L \rightarrow (B; (\text{DEFINE}\not{\emptyset} B \not{\#}); (\text{READ}\not{\emptyset} B \not{\#});\ (\varphi),\ (\varphi))$
11. $L \rightarrow (C; (\text{DEFINE}\not{\emptyset} C \not{\#}); (\text{READ}\not{\emptyset} C \not{\#});\ (\varphi),\ (\varphi))$
12. $L \rightarrow (D; (\text{DEFINE}\not{\emptyset} D \not{\#}); (\text{READ}\not{\emptyset} D \not{\#});\ (\varphi),\ (\varphi))$

---

1. To avoid ambiguity, we shall here represent an n-translation form as $(\alpha_1; (\beta_1; \pi_1); \ldots; (\beta_n; \pi_n))$.

13. L → (E;(DEFINE/E=#);(READ/E=#);(φ); (φ))

14. E → (L_2;(L_2);(L_2);(L_2);(LOAD/L_2=#))

15. E → (M;M;M;M;M)

16. E → (E–L_2;(EL_2);(EL_2);(EL_2);(ESUB/L_2=#))

17. E → (E–M;(EM);(EM);(M ST//T+m=# m=m+1 E);
(E SUB/T+n=# n = n+1))

18. M → (L_1*L_1;(L_1L_1);(L_1L_1);(LOAD/L_1=#MPY/L_1=#;[1,2]);
(L_1L_1; [1, 2]))

19. M → (M*L_1; (ML_1);(ML_1);(M MPY/L_1=#);(ML_1))

20. L_1 → (A; φ; φ; A; φ)

21. L_1 → (B; φ; φ; B; φ)

22. L_1 → (C; φ; φ; C; φ)

23. L_1 → (D; φ; φ; D; φ)

24. L_1 → (E; φ; φ; E; φ)

25. L_2 → (A; φ; φ; φ; A)

26. L_2 → (B; φ; φ; φ; B)

27. L_2 → (C; φ; φ; φ; C)

28. L_2 → (D; φ; φ; φ; D)

29. L_2 → (E; φ; φ; φ; E)

30. S_2 → (PUNCH/L_2;(L_2);(L_2);(L_2);(PUNCH/L_2=#).

APPENDIX B

n - PUSHDOWN ASSEMBLERS

In Chapter 2, we introduced the concept of pushdown assemblers. Specifically, we described two classes of machines called type A pushdown assemblers (APA) and type B pushdown assemblers (BPA). We also stated various results (Theorem 1.31 - 1.37) which relate pushdown assemblers with syntax directed translations. In this appendix, we shall extend the features of these machines so that they can characterise n-syntax directed translations. In Chapter 4 (4.1), we gave some conjectures relating n-pushdown assemblers with nSDT's.

Here again, we shall talk of two classes of machines: n-type A pushdown assembler (nAPA) and n-type B pushdown assembler (nBPA). We shall describe these machines by discussing the additional features that these have over pushdown automata (discussed in Chapter 1). Let us describe nAPA first:

Here, associated with each tape symbol on the pushdown list, are n sets of k($\geq$ 1) passive registers. A tape symbol together with its registers will be called 'level'. Each register can be empty or hold a string of output symbols. Such a situation is shown in Fig. B.1, where k = 2, n = 2 and the pushdown list contains the tape symbols CBA, with C on the top of the list.

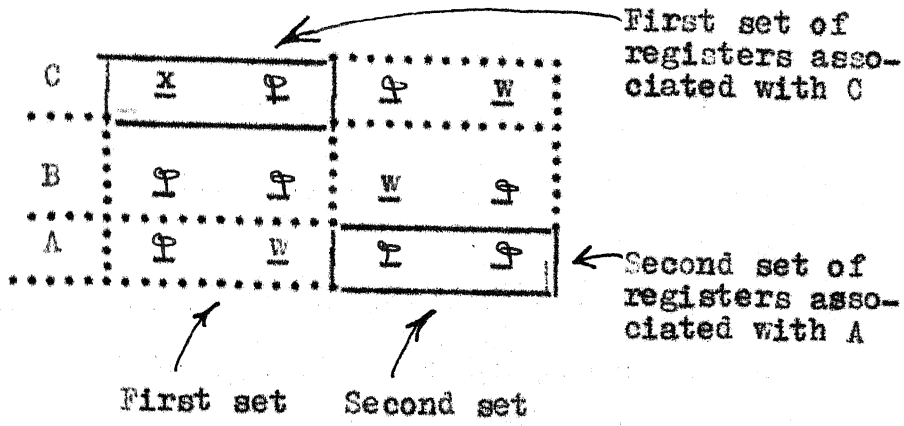


Figure B.1

Associated with A are two sets of registers. In the first set, the first register is empty (φ indicates an empty register) and the second register is holding string w. In the second set, both the registers are empty. Both registers in the first set, associated with B are empty while in the second set, the first register contains string w and the second register is empty. In the first set of registers associated with C, the first register contains string x and the second register is empty while in the second set, the first register is empty and the second register contains string w.

Suppose C, at the top of the list, is replaced by string Ed. Symbol D would replace C in Fig. B.1 and E would appear above D with empty registers. The result appears in Figure B.2.

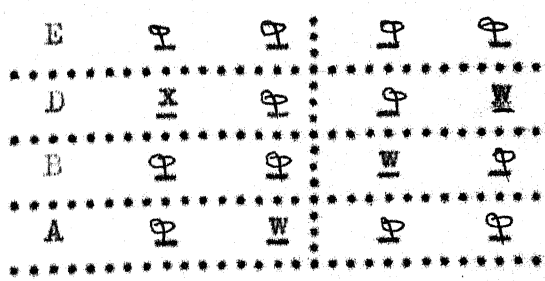


Figure B.2

The nAPA can write a finite string in any empty register at the highest level. For example, it could write y and z in the first and second registers of the first set and z in the first register of the second set at the top level, leaving the situation of Figure B.3.

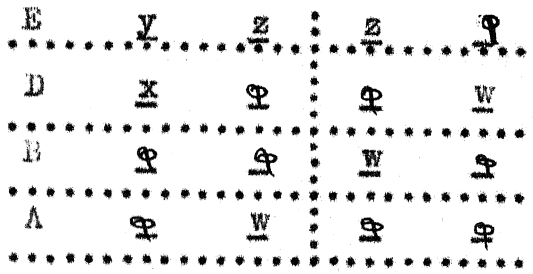


Figure B.3

If the top tape symbol is erased, the contents of its registers in each set are concatenated in order. If a register is empty, it is treated as though it contained ϵ (the null string). The resulting n strings, (i-th string being the result of concatenating the contents of registers in the i-th set, in order) then 'wait' at the top of the list, the i-th 'waiting string', $1 \leq i \leq n$, to be placed on the next move in some empty register of the i-th set (from left). If E in Figure B.3 were erased, yz and z would be passed down to the level below and would appear temporarily to the left of D as in Figure B.4.

Next, yz is placed in an empty register of the first set and z in an empty register of the second set, at the top level.

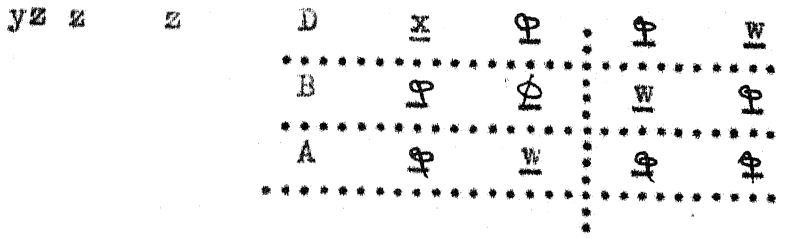


Figure B.4

In this case, yz must be put in the second register of the first set and z must be placed in the first register of the second set. If the nAPA tries to write into a register which is not empty, it 'jains' and can make no further moves. Figure B.4 thus should become Figure B.5.

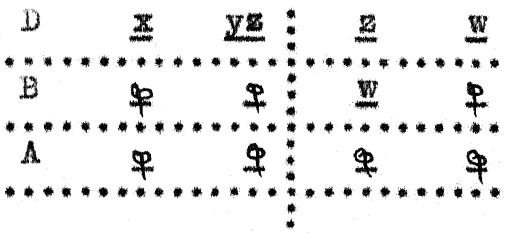


Figure B.5

If next, D were erased and the strings resulting from the concatenation of registers in its first and second sets were stored in the first register and second register of the first and second sets respectively, the list would be as in Figure B.6.

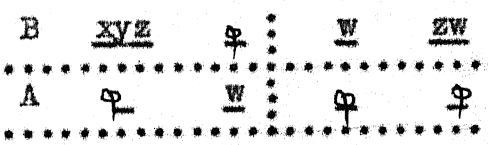


Figure B.6

Finally, suppose that on successive moves, B is erased, xyz stored in the first register of the first set and wzw stored in the first register of the second set, of the next level, and A is erased. The resulting strings xyzw (from the first set) and wzw (from the second set) would have no place to go. The concatenation of the two strings in order will be deemed output of the nAPA; xyzwwzw would be deemed a translation of whatever input string caused the sequence of moves, the terminal portion of which we have been describing.

We shall now give a formal notation incorporating the ideas we have been using. A k-register n-type A pushdown assembler (n(k)APA) is defined as the system:

$$M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0),$$

where Q, Σ, Δ, and Γ are finite sets of states, input symbols, output symbols and tape symbols, respectively. q_0, in Q, is the start state. Z_0, in Γ, is the start symbol. λ, μ and ν are mappings which indicate the allowable moves of M. λ controls changes of tape symbols (not register contents) on the pushdown list. μ controls the entry of finite length output strings into registers. ν controls the insertion into registers of output strings which have been displaced temporarily by the erasure of the top symbol on the pushdown list (as in Figure B.4).

λ is a mapping from $Q \times (\Sigma \cup \{e\}) \times \Gamma$ to the finite subsets of $Q \times \Gamma^*$.

If $\lambda(p, a, Z)$ contains (q, ϵ), then if the topmost tape symbol is Z, M can erase the top level, transfer from state p to q and use input a (possibly ϵ). If $\lambda(p, a, Z)$ contains $(q, X_1X_2 \ldots X_m)$, $(m \geq 1)$, M can replace Z by X_m at the top level, then grow new levels on the top of the pushdown list with empty registers and tape symbols $X_1, X_2, \ldots, X_{m-1}$, in order from the top.

μ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times (\Delta_1^* \times \{1,2, \ldots, k\}) \times \ldots \times (\Delta_n^* \times \{1,2, \ldots, k\})$ where Δ_i, $1 \leq i \leq n = \Delta$.

If $\mu(p, a, Z)$ contains $(q, (x_1, i_1), \ldots, (x_n, i_n))$ then when Z is the top tape symbol, M can use input a, transfer from state p to q and write output string x_j in the i_j-th register (from left) of the j-th set of registers of the top level, provided that register is empty.

ν is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the subsets of $Q \times \{1,2, \ldots, k\}_1 \times \{1, 2, \ldots, k\}_2 \times \ldots \times \{1,2, \ldots, k\}_n$[1].

If $\nu(p, a, Z)$ contains $(q, i_1, i_2, \ldots, i_n)$, and if Z is the top tape symbol and the level above has just been erased, M can use input a, transfer from state p to q and store the j-th string passed from the level above, in the i_j-th register of the j-th set of registers of the current top level. Again, that register must be empty.

A <u>configuration</u> of M is denoted (q, α), where q is in Q, and α is a string either of the form $Z_1t_1Z_2t_2 \ldots Z_mt_m$ or of the form $[w_1, w_2, \ldots, w_n]\, Z_1t_1Z_2t_2 \ldots Z_mt_m$, where $Z_1, Z_2, \ldots, Z_m$ are

1. Here, $\{1,2, \ldots, k\}_j$ indicates the j-th set $\{1,2,\ldots, k\}$, $1 \leq j \leq n$.

in Γ, $t_i = ((t_{i1}), (t_{i2}), \ldots, (t_{in}))$, $(1 \leq i \leq m)$, $t_{i1}, t_{i2}, \ldots, t_{im}$ being k-tuples of elements in $\Delta^* \cup \{\varphi\}$, and w_i, $1 \leq i \leq n$, is in Δ^*. t_{ij}, $1 \leq i \leq m$, $1 \leq j \leq n$, represents the contents of the k registers in the j-th set of registers associated with Z_i. φ denotes an empty register. The string α is prefixed by $[w_1, w_2, \ldots, w_n]$ if a tape symbol has been erased on the previous move, a condition akin to that of Figure B.4.

For any q in Q, a in $\Sigma \cup \{\epsilon\}$, Z in Γ and n tuple t, suppose $\lambda(q, a, Z)$ contains $(p, Z_1 Z_2 \ldots Z_m)$, $m \geq 1$, then we may write $a : (q, Zt\alpha) \vdash_M (p, Z_1 t_o Z_2 t_o \ldots Z_{m-1} t_o Z_m t \alpha)$. If $m = 1$, then we may write,

$a : (q, Zt\alpha) \vdash_M (p, Z_1 t \alpha)$. If $\lambda(q, a, Z)$ contains (p, ϵ), then we may write,

$$a : (q, Z((w_{11}, \ldots, w_{1k}), \ldots, (w_{n1}, \ldots, w_{nk}))\alpha) \vdash_M (p, [w_{11} \ldots w_{1k}, \ldots, w_{n1} \ldots w_{nk}]\alpha).$$

Here, $w_{i1} w_{i2} \ldots w_{ik}$ is the concatenation of $w_{i1}, w_{i2}, \ldots, w_{ik}$, $1 \leq i \leq n$, with φ taken to be ϵ.

Suppose $\lambda(q, a, Z)$ contains $(p, i_1, i_2, \ldots, i_n)$, then we may write,

$$a : (q, [w_1, \ldots, w_n] Z((x_{11}, \ldots x_{1k}), \ldots, (x_{n1}, \ldots, x_{nk}))\alpha) \vdash_M (p, Z((x_{11}, \ldots, x_{i_1-1}, w_1, x_{i_1+1}, \ldots, x_{ik}), \ldots, (x_{n1}, \ldots, x_{i_n-1}, w_n, x_{i_n+1}, \ldots, x_{nk}))\alpha),$$

again, provided x_{i_j}, $1 \leq j \leq n = \varphi$.

The symbol $\vdash_M^*$ is defined as follows:

For any configuration (q, α), $\epsilon : (q, \alpha) \vdash_M^* (q, \alpha)$. For w in Σ^* and a in $\Sigma \cup \{\epsilon\}$, if $w : (q_1, \alpha_1) \vdash_M^* (q_2, \alpha_2)$ and $a : (q_2, \alpha_2) \vdash_M (q_3, \alpha_3)$, then $wa : (q_1, \alpha_1) \vdash_M^* (q_3, \alpha_3)$. The <u>translation defined by</u> M, denoted $\tau(M)$ is:

$\{(w, x) \mid w: (q, Z_0 t_0) \vdash_M^* (q, [x_1, x_2, \ldots, x_n])$ and

$x = x_1 x_2 \ldots x_n$, for some q in $Q\}$.

The n-type B pushdown assembler (nBPA) differs from the nAPA as follows: The nBPA always concatenates strings to the left or right of the present contents of a register. The nBPA is thus allowed to write into a non-empty register.

Formally, a <u>k-register n-type B pushdown assembler</u> (n(k)BPA) is defined as:

$$M = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0),$$

where all symbols (within parenthesis) have the same meaning as for the n(k) APA except:

(i) μ maps $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times (\Delta_1^* \times \{1, \ldots, k\} \times \{L, R\}) \times \ldots \times (\Delta_n^* \times \{1, \ldots, k\} \times \{L, R\})$, where $\Delta_i = \Delta$, $1 \leq i \leq n$.

If $\mu(p, a, Z)$ contains $(q, (x_1, i_1, D_1), \ldots, (x_n, i_n, D_n))$, then under appropriate conditions M can concatenate x_j to the left or right end of the contents of the i_j-th register of the j-th set at the top level, depending on whether D_j = L or R, respectively.

(ii) $\sqrt{}$ maps $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the subsets of $Q \times (\{1,2,\ldots,k\}_1 \times \{L, R\}) \times \ldots \times (\{1,2,\ldots,k\}_n \times \{L, R\})$.

If $\sqrt{}(p,a,Z)$ contains $(q, (i_1, D_1), (i_2, D_2), \ldots, (i_n, D_n))$, then under appropriate conditions, M can concatenate the j-th string (from left) passed from the level above, to the left or right of the current contents of i_j-th register of the j-th set at the top level, depending on whether $D_j = L$ or R, respectively.

A <u>configuration</u> of M is denoted (q, α), as for the n(k)APA. The n-tuples in string α are composed of elements in Δ^* rather than $\Delta^* \cup \{\phi\}$. For a n(k)BPA, ϵ instead of ϕ, can be used for an empty register.

The symbol $\vdash_M$ is defined by:

(i) If $\lambda(q, a, Z)$ contains $(p, Z_1 Z_2 \ldots Z_m)$, $(m > 1)$, then
$a : (q, Zt\alpha) \vdash_M (p, Z_1 t_o Z_2 t_o \ldots Z_{m-1} t_o Z_m t\alpha)$,

(ii) If $\lambda(q, a, Z)$ contains (p, Y), then
$a : (q, Zt\alpha) \vdash_M (p, Yt\alpha)$,

(iii) If $\lambda(q, a, Z)$ contains (p, ϵ), then
$a : (q, Z((w_{11}, \ldots, w_{1k}), \ldots, (w_{n1}, \ldots, w_{nk}))\alpha)$
$\vdash_M (p, [w_{11} \ldots w_{1k}, \ldots, w_{n1} \ldots w_{nk}]\alpha)$.

(iv) If $\sqrt{}(q, a, Z)$ contains $(p, (i_1, D_1), \ldots, (i_n, D_n))$, then
$a : (q, [w_1, \ldots, w_n] Z((x_{11}, \ldots, x_{1k}), \ldots, (x_{n1}, \ldots, x_{nk}))\alpha)$
$\vdash_M (p, Z((y_{11}, \ldots, y_{1k}), \ldots, (y_{n1}, \ldots, y_{nk}))\alpha)$,

where,

$$y_{1j} = x_{1j} \quad \text{for } j \neq i_1$$
$$y_{2j} = x_{2j} \quad \text{for } j \neq i_2$$
$$\vdots$$
$$y_{nj} = x_{nj} \quad \text{for } j \neq i_n.$$

$y_{ki_k} = w\, x_{ki_k}$ or $x_{ki_k}\, w$, as $D_k = L$ or R, $1 \leq k \leq n$,

respectively.

(v) If $\mu(q, a, Z)$ contains $(p, (w_1, i_1, D_1), \ldots, (w_n, i_n, D_n))$, then $a : (q, Z((x_{11}, \ldots, x_{1k}), \ldots, (x_{n1}, \ldots, x_{nk}))\, \alpha)$

$$\vdash_M (p, Z((y_{11}, \ldots, y_{1k}), \ldots, (y_{n1}, \ldots, y_{nk}))\, \alpha),$$

where

$$y_{1j} = x_{1j}, \quad j \neq i_1$$
$$y_{2j} = x_{2j}, \quad j \neq i_2$$
$$\vdots$$
$$y_{nj} = x_{nj}, \quad j \neq i_n$$

$y_{ki_k} = w x_{ki_k}$ or $x_{ki_k}\, w$, as $D_k = L$ or R, $1 \leq k \leq n$, respectively.

The symbol $\vdash_M^*$ as defined from $\vdash_M$ exactly as for n(k)APA.

The translation defined by M, denoted $\tau(M)$, is:

$$\{(w, x) \mid w : (q_0, Z_0 t_0) \vdash_M^* (q, [x_1, \ldots, x_n]) \text{ and } x = x_1 \ldots x_n, \text{ for some } q \text{ in } Q\}.$$

If M is an n(k)BPA, then $\tau(M)$ will be called a B-translation of order k.